2010 Second International Conference on Computer Engineering and Applications

An Improved Pipelined Processor Architecture Eliminating Branch and Jump Penalty

Md. Raqibul Hasan
Bangladesh University of Engineering and Technology
raqib_cse@yahoo.com

M. Sohel Rahman
Bangladesh University of Engineering and Technology
msrahman@cse.buet.ac.bd

Masud Hasan
Bangladesh University of Engineering and Technology
masudhasan@cse.buet.ac.bd

Md. Mahmudul Hasan
East West University, Bangladesh.
mmhh@edubd.edu

M. Ameer Ali
East West University, Bangladesh.
maa@ewubd.edu

***Abstract*— Control dependencies are one of the major limitations to increase the performance of pipelined processors. This paper deals with eliminating penalties in pipelined processor. We present our discussion in the light of MIPS pipelined processor architecture. Here we present an improved pipelined processor architecture eliminating branch and jump penalty. In the proposed architecture CPI for branch and jump instruction is less than that of MIPS architecture. We also have shown the design of the required cache memory cell for the improved architecture.**

I. Introduction

Most modern microprocessor designs use pipelining [1,2] to significantly increase performance. Pipeline is an implementation technique in which multiple instructions are overlapped in execution. In pipelining the expectation is to fetch instruction[1] in each clock cycle.

The operational steps in pipelined architecture are called pipeline stages. In MIPS pipelined architecture [2,3] there are five pipeline stages, including instruction fetch (IF), instruction decode (ID), execution (EX), memory access (MEM) and result writeback (WB). It is important that the operation time of each pipeline stage is almost identical, since the rate at which instructions flow through the pipeline is limited by the slowest pipeline stage. To retain the operand and control signals of an individual instruction for its other stages, the operand and control signals are saved in a register called pipeline register. In MIPS architecture we have four such registers.

IF/ID is the pipeline register between instruction fetch (IF) and instruction decode (ID) stage.

Similarly the remaining three pipeline registers are named.

Recall that the typical operational steps in the execution of an instruction. Let the instruction pointed by PC (program counter) is fetched in the i-th clock cycle. Simultaneously at the end of that clock cycle PC is incremented to point the next instruction. In the $i+1$-th clock cycle the instruction is in ID pipeline stage. Here the register values that will be used in EX stage are accessed from register file [2]. In the next clock cycle i.e $i+2$-th the instruction is in EX stage and required operation in the ALU (arithmetic and logic unit) is performed. If the instruction requires to access memory, it is done in the $i+3$-th clock cycle. $i+4$-th is the last operational clock cycle for that instruction. In this clock cycle result of arithmetic operation or accessed data from memory is written in destination register if required.

A branch is a point in a computer program where the flow of control is altered. A branch may be taken or not taken. If a branch is not taken, the flow of control is unchanged and the next instruction to be executed is the instruction immediately following the current instruction in memory; if taken, the next instruction to be executed is an instruction at some other place in memory (branch destination instruction). Branch instructions can reduce the performance of pipelines by interrupting the steady flow of instructions into the pipeline. To execute a branch instruction, the processor must decide whether the branch is taken, calculate the branch's destination address. Calculating destination address and checking branch condition may be conducted in parallel. But the destination instruction can be fetched only after calculating the destination address if branch target buffer (BTB) is not used. Branch hurt performance because these tasks can not be performed in one clock cycle.There are some techniques for reducing branch penalty [4]. For further improved performance now branch prediction [5,6,7] is used. If the prediction is correct there is no penalty. But if prediction is incorrect there is a penalty of one clock cycle.

In this paper we are concerned about improving performance of branch instruction. Remaining improved features will come as the consequence of improving the former feature. The rest of the sections are organized as follows. In Section 2 we briefly review the execution of branch instruction in current MIPS architecture. In section 3 we have presented our proposed architecture eliminating branch and jump penalty. Finally, we briefly conclude in Section 4.

[1]Reading the instruction from memory and placing in the IF/ID pipeline register. The notation IF/ID is defined later.

[2]A state element that consists of a set of registers that can be read or written by supplying a register number to be accessed.

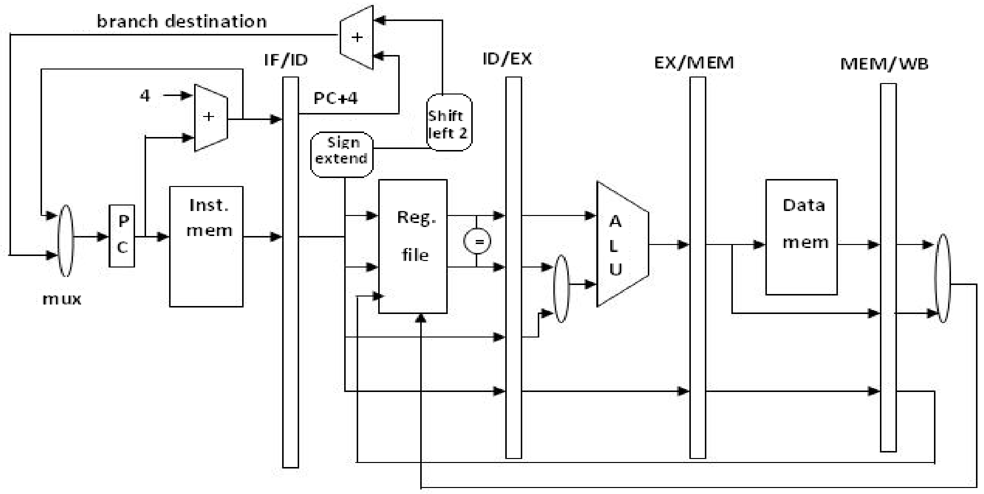

Fig. 1. The MIPS architecture.

II. Execution of Branch Instruction in MIPS

In MIPS architecture length of each instruction is 32 bit (4 byte). In branch instruction there is a offset of 16 bit and id[3] (identification number) of two registers which determine whether the branch will be taken or not taken. Here is the meaning of each names of the fields in the machine code of branch instruction.

op	rs	rt	offset
6 bit	5 bit	5 bit	16 bit

Fig. 2. Machine code format of branch instruction in MIPS.

op: Basic operation of the instruction, traditionally called opcode.

rs: The first register source operand.

rt: The second register source operand.

offset: Distance of the destination instruction in number of instruction from the branch instruction. As each instruction length is 4 byte, here branch range is 4 times larger than that of if distance of destination instruction is measured in number of byte.

Branch destination address is calculated as follows

PC_{new} = PC(of branch inst.) + 4 + offset (sign extended, 2 bit left shifted)

There are two types of branch instruction in MIPS e.g *beq* and *bne*. In case of *beq* branch is taken if *rs* and *rt* registers are equal, otherwise not taken. On the other hand in case of *bne* branch is taken if *rs* and *rt* are not equal, otherwise not taken. All the executional steps of these two instructions are almost similar except role of some selection signals (zero flag). When executing a branch instruction operations performed in each pipeline stages are as follows:

IF: Branch instruction is fetched from the instruction memory and the PC is incremented.

ID: Two register *rs* and *rt* are read from register file.

EX: The ALU performs a subtract on the data values read from the register file. The value of PC+4 is added to the sign-extended, lower 16 bit of the instruction (*offset*) shifted left by two; the result is the branch destination address.

MEM: The zero flag from the ALU is used to decide whether PC+4 or destination address will be stored in PC.

So if a branch instruction is fetched in i-th clock cycle, the next instruction can be fetched confidently in $i+4$-th clock cycle. As a result there may be a penalty of three clock cycle i.e three wrong instruction may be fetched which have to flush[4].

Here the next PC for a branch is selected in the MEM stage. To obtain better performance branch execution is moved earlier in the pipeline. Moving the branch decision up requires two actions to occur earlier: computing the branch destination address and evaluating the branch decision. Branch address is calculated in ID stage as value of PC+4 is already present in IF/ID pipeline register. Equality of two register is tested by first exclusive ORing their respective bits and then ORing all the results. This is done in the ID stage after accessing register values from register file. As a result if a branch instruction is fetched in i-th clock cycle, next instruction can be fetched confidently in $i+2$-th clock cycle. So the branch penalty is reduced to one clock cycle.

The branch target buffer (BTB) [7,8] is used to reduce the performance penalty of branches by predicting the path of the branch and caching information about the branch. Branch target buffer (BTB) stores the destination address of the last target location to avoid recomputing.

If a branch instruction is fetched and prediction for that instruction is taken, then the destination instruction is fetched in the next clock cycle. If the corresponding prediction is not taken, then the next sequential instruction pointed by PC is fetched from memory in the next clock cycle. If the

[3]Can be treated as address of a register.

[4]To discard instructions in pipeline, usually due to unexpected event.

prediction is wrong, then the incorrect instruction is deleted which causes a loss of one clock cycle. Assume that the fraction of branch instructions predicted incorrectly is P i.e $P\times 100\%$ branch instructions are predicted incorrectly. Then in MIPS architecture CPI (clock per instruction) for branch instruction using branch prediction and branch target buffer is $1+P$.

Delayed branch [9] is a simple solution for branch cost. A delayed branch always executes the following instruction, but the second instruction following the branch will be affected by the branch. Compilers and assemblers try to place an instruction that always executes after the branch. But most of the time delayed branch can not provide good performance.

III. The Proposed Architecture

This section presents the proposed architecture eliminating branch and jump penalty. Recall that in MIPS architecture, if the branch is taken the calculation of the target address require one clock cycle. PC_{new} = PC_{old} + 4 + *offset* (sign extended & 2 bit left shifted). To reduce the cost of destination address calculation we generate the destination address in a different way. Here we shall consider segmented memory. The address of a memory location is determined by corresponding segment number and segment offset. We have modified the machine code format of branch instruction as follows.

op	rs	rt	selection	segment offset
6 bit	5 bit	5 bit	2 bit	14 bit

Fig. 3. Machine code format of branch instruction in the proposed architecture.

Least significant two bit of each instruction address is '00' as each instruction is 4 byte. In branch instruction *segment offset* field is of 14 bit. If the 14 bit offset is left shifted two bit we shall get 16 bit number. So most significant 16 bit of the 32 bit address [31:0] is chosen for the segment number and least significant 16 bit for segment offset. Let current segment of PC is denoted by *PCseg*. Then *PCseg-1* and *PCseg+1* are the upper and lower segments of *PCseg* respectively. *PCseg±2* will be *PCseg-2* if current PC is in the upper half portion of *PCseg* and *PCseg+2* otherwise. For the branch destination calculation we shall get the *segment offset* from the instruction and for segment number we shall take any one of *PCseg*, *PCseg-1*, *PCseg+1* or *PCseg±2* according to the *selection* bit in the branch instruction.

Branch destination address = [*PCseg-1* |[5] *PCseg* | *PCseg+1* | *PCseg±2*] o [6] [14 bit *offset* shifted left by two bit].

We shall keep the value of *PCseg-1*, *PCseg*, *PCseg+1* and *PCseg±2* in four registers. If current segment of PC is changed for increment of PC or for update of PC by branch or jump destination address *PCseg-1*, *PCseg*, *PCseg+1* and *PCseg±2* are updated. *PCseg±2* is also updated if PC change position from upper half portion of *PCseg* to lower half portion or vice versa. The time required to calculate destination address is the time required to select a segment from four segments according to the *selection* bit i.e the propagation delay of a 4 to 1 mux which is very smaller than that of a 32 bit adder used in MIPS. Here our candidate segments are *PCseg-1*, *PCseg*, *PCseg+1* and *PCseg±2* as we have two *selection* bit.

[5]This symbol represents OR.

[6]This symbol represents concatenation.

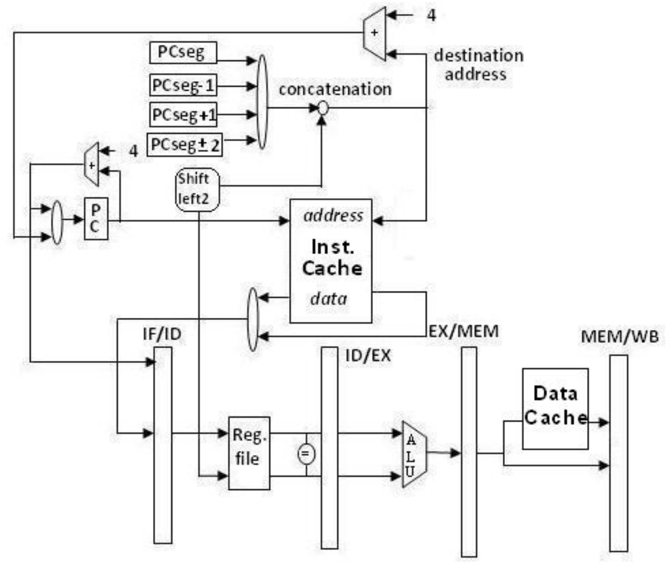

Fig. 4. The Proposed architecture.

In Figure 5 selection signal for *mux1* type mux depends on branch condition and selection signal for *mux2* (*mux3*) type mux comes from 15-th bit of PC+4 (branch destination + 4), where bits of 32 bit address are numbered as [31:0].

If a branch instruction is fetched in the i-th clock cycle, in the $i+1$-th clock cycle we shall try to read two instruction simultaneously, one addressed by PC and another addressed by destination address. Then we shall load one of the two instructions in the IF/ID pipeline register depending on branch condition. Here we require memory (instruction cache) that is capable to read two location simultaneously addressed by two different addresses. First we shall discuss how assembler will be modified to cope with the new destination address calculation and later about required new memory. Here we are not discussing data forwarding and exception handling units because these units will remain same as in MIPS.

A. Modification of Assembler

Segment offset represent the position of the instruction in a particular segment. To determine the machine code for the assembly instruction *beq $1,$2,target* the assembler has to determine the segment offset of the target instruction and have to determine whether the segment number of target instruction is *PCseg*, *PCseg-1*, *PCseg+1* or *PCseg±2*. Obviously it is possible by implementing the assembler

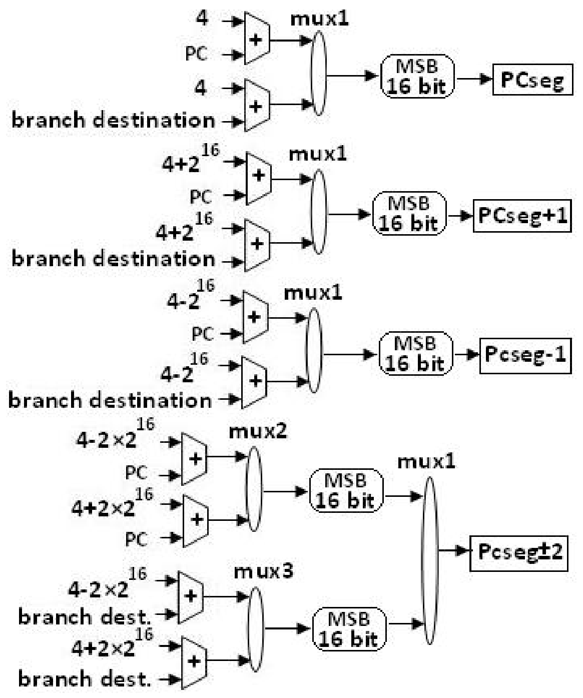

Fig. 5. Circuit for updating *PCseg*, *PCseg+1*, *PCseg-1* and *PCseg*±*2*.

appropriately. Here we have to restrict the loader[7] such that it copies the instructions of a program starting at the beginning of a segment. So the assembler can determine the *segment offset* of the *target* address and it is also determinable whether the *target* address is in the segment *PCseg*, *PCseg-1*, *PCseg+1* or *PCseg*±*2*. The *selection* field in the machine code of branch instruction will be filled on the basis of the segment number of the *targer* address according to the table in Figure 6. Other remaining fields will be handled as like

segment number	selection
PCseg	00
PCseg-1	01
PCseg+1	10
PCseg±*2*	11

Fig. 6. Selection bit in machine code of branch instruction.

in MIPS.

B. Modified Instuction Cache Cell

In the conventional memory there is one address line and a corresponding data line i.e at a time one address can be provided as input. Pipelined architectures use two memories, one for instruction and another for data. The use of two memory is obvious. If in a particular clock cycle instruction in the MEM stage is load or store then it require memory access for data. At the same time fetching new instruction in the IF stage require memory read. Without two memory the pipeline could have a structural hazard[8]. So the use of two memory is obvious. As now we are using hierarchical memory we require two cache memory, one for instruction and another for data. In this case only one RAM (main memory) will provide our requirement. Moreover in most of the systems separate cache for instruction and data is used. Because with unified cache, a program that is data-intensive quickly fill the cache, allowing little room for instructions. This slows the execution speed of the processor.

In the proposed architecture we are going to use an instruction cache that is capable to read two location simultaneously addressed by two different addresses. In the modified instruction cache we have two independent sets of address and data lines e.g *address line1*, corresponding *data line1* and *address line2*, corresponding *data line2*. We can give two independent address in the two address lines simultaneously and corresponding contents will be available in the corresponding data lines with the same propagation delay of single address and data line cache.

2-D memory [10] is widely used in most of the applications. In 2-D memory there are two selection lines X (row selector) and Y (column selector) generated by address decoders (row decoder & column decoder) are used to select a memory cell [11]. In such memory n-bit address is divided into two parts, one part is given as the input to the row decoder and another part is given to the column decoder.

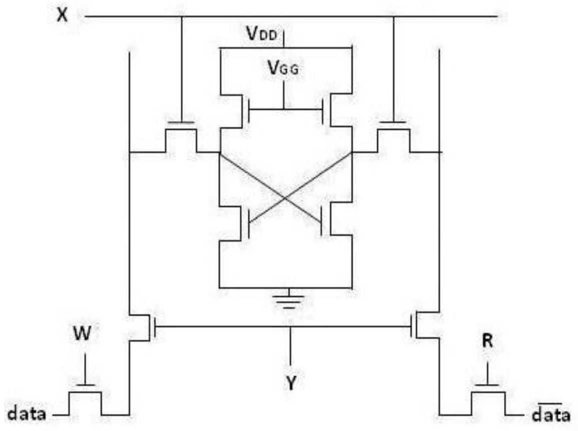

Fig. 7. Six transistor static memory cell.

In the modified cache memory we have two sets of selection lines X1,Y1 and X2,Y2 generated from row, column decoder1 and row, column decoder2. Row, column decoder1 for *address line1* and row, column decoder2 for *address line2*. In Figure 8 MOS transistor diagram of a modified cache memory cell is given. Here we require one additional MOS transistor (T1) for each memory cell (bit). Transistor T2 is additional for each column.

[7]A system program that places an object program in main memory so that it is ready to execute.

[8]An occurrence in which a planned instruction can not execute in the proper clock cycle because the hardware cannot support the combination of instructions that are set to execute in the given clock cycle.

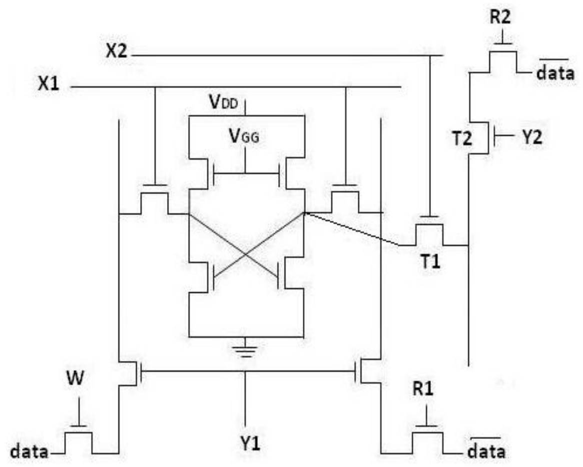

Fig. 8. Modified memory cell.

C. Eliminating Branch Penalty

Assume that a branch instruction is fetched in the i-th clock cycle. In the $i+1$-th clock cycle we shall read two instruction simultaneously, one addressed by PC through *address* and *data line1* and another one addressed by branch destination address (computation of which require very small time) through *address* and *data line2*. Then at the end of the $i+1$-th clock cycle we shall load one of the two instructions in the IF/ID pipeline register depending on branch condition. Thus the branch instruction has no penalty. Here we want to denote that we no longer require branch predictor and branch target buffer.

D. Eliminating Jump Instruction Cost

There is also improvement for jump instruction.

jump destination= [most significant 4 bit of PC + 4] o [2 bit left shifted least significant 26 bit of the jump instruction]

In MIPS architecture for jump instruction first PC is updated with the jump target address and in the following clock cycle target instruction is fetched. So CPI for jump instruction is 2.

As we require no time to generate the jump address as like branch destination address. If a jump instruction is fetched in a particular clock cycle then in the following clock cycle through *address* and *data line1* we shall access instruction addressed by jump destination not by PC. In that clock PC will be updated with jump destination + 4.

IV. Conclusion

In this paper we have presented an architecture that eliminates branch and jump penalty completely. In the proposed architecture the CPI for branch instruction is less than that of MIPS architecture and it is 1 instead of 1.25 if P=0.25 is assumed. CPI for jump instruction is 1 instead of 2. Here we no longer require branch predictor and branch target buffer.

References

[1] P.M. Kogge, *The Architecture of Pipelined Computers*, McGraw-Hill, 1981

[2] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design*, Elsevier Inc., 2005

[3] G. Kane, *MIPS RISC Architecture*, Prentice Hall, 1989

[4] S. McFarling, J. Henneseey*Reducing the cost of branches*, Proceedings of the 13th annual international symposium on Computer architecture (ISCA '86) Pages: 396 - 403

[5] J.E. Smith, *A Study of Branch Prediction Strategies*,Proc. Eighth Symp. Computer Architecture, May 1981, pp. 135-148

[6] J.K.F. Lee and A.J. Smith, *Branch Prediction Strategies and Branch Target Buffer Design*, Computer, Jan. 1984, Volume 17,pp. 6-22

[7] R. Nair, *Optimal 2-Bit Branch Predictors*, Trans. Computers, volume 44, p. 698-702 (1995).

[8] C. Perleberg and A. J. Smith, *Branch Target Buffer Design and Optimization*, IEEE Trans. Computers, volume 42, p. 396-412 (1993).

[9] T. R. Gross , John L. Hennessy, *Optimizing delayed branches*, Proceedings of the 15th annual workshop on Microprogramming, p.114-120, October 05-07, 1982, Palo Alto, California, United States

[10] Luecke, G., J.P. Mize and W.N. Carr, *Semiconductor Memory Design and Applications*, chap. 3, McGraw-Hill Book Company, New York, 1973

[11] Terman, L.M., *MOSFET memory Circuits*, Proc. IEEE, vol. 59, no. 7,pp.1044-1059, July 1971