PROVING THE CORRECTNESS OF PROCESSORS WITH DELAYED BRANCH USING DELAYED PC

Silvia M. Mueller, Wolfgang J. Paul, and Daniel Kroening

Abstract:

We show that the programming model of delayed branch is equivalent to what we call *delayed PC*: *all* instruction fetches are delayed by one instruction, not just taken branches. This leads to a very simple new implementation of the delayed branch mechanism. We then prove the correctness of a pipelined machine with delayed PC.

INTRODUCTION

Machine verified correctness proofs for (almost) entire processors have been produced for sequential machines [1], for pipelined machines [2, 3, 4, 5, 6] and for machines with out of order execution [7, 6, 8, 9]. In all non sequential designs cited above either a branch-not-taken strategy is applied or the following actions are performed in a *single cycle*: i) the evaluation of the condition of branch instructions ii) the next PC computation iii) the fetch of the next instruction.

In real machine these three actions are usually performed in two or more cycles in order to reduce cycle time. This does not remain invisible to the programmer: taken *branches* are *delayed* by one or more instructions. The delayed branch semantics is, for example, used in the MIPS [10], the SPARC [11] and the PA-RISC [12] instruction set.

In this paper we show that the programming model of delayed branch is equivalent to what we call *delayed PC*: *all* instruction fetches are delayed by one instruction, not only taken branches. This leads to a very simple new implementation of the delayed branch mechanism. We then prove the correctness of a pipelined machine with delayed PC. Parts of the proof have been verified by machine already.

The paper is organized in the following way: In the next section, we formally define the semantics of a DLX machine [13] with delayed branch and delayed PC, and we show that they are equivalent. We then describe a *sequential* machine DLX_σ with the following features: i) delayed PC, ii) pipelined data

paths with a 5 stage pipeline, iii) the pipeline stages are clocked in a round robin fashion. In the last section we turn the sequential machine DLX_σ into a pipelined machine DLX_π by only 2 changes: i) the delayed PC of the sequential machine is bypassed, ii) the clocking of the pipeline stages is modified. We then show that the pipelined machine simulates the sequential machine.

DELAYED BRANCH AND DELAYED PC

We consider sequences $I = I_0, I_1, ...$ of DLX instructions started after reset. For registers R and instructions I_i we denote by R_i the value of register R after sequential execution of instruction I_i. By R_{-1} we denote the initial value of R before the execution of I_0. Observe that for sequential machines instruction I_i is fetched from memory address PC_{i-1}.

A (sequential) semantic of delayed branch requires the introduction of state variables which memorize, whether previous instructions were taken branches (or jumps), and memorize the branch target of branches/jumps. We use the variables $bjtaken$ and $btarget$. If I_i is a relative branch/jump with immediate constant imm_i or an absolute jump with operand $RS1_{i-1}$, then for machines with delayed branch the branch target is

$$btarget_i = \begin{cases} RS1_{i-1} & \text{for absolute jumps} \\ PC_{i-1} + 4 + imm_i & \text{for relative branch/jumps} \end{cases}$$

Observe that the addition of 4 is an artifact. The variable $bjtaken_i = 1$ indicates that instruction I_i is a jump or a taken branch. The machine is initialized with

$$PC_{-1} = 0 \quad \text{and} \quad bjtaken_{-1} = 0.$$

The delayed branch mechanism is specified by

$$PC_{i+1} = \begin{cases} btarget_i & \text{if} \quad bjtaken_i = 1 \\ PC_i + 4 & \text{otherwise} \end{cases}$$

and by the requirement that delay slots do not contain branch instructions.

The delayed PC mechanism uses a program counter PC' and its delayed version DPC which is used for fetching instructions. They are initialized with

$$DPC_{-1} = 0 \quad \text{and} \quad PC'_{-1} = 4.$$

The computation of the next PC' is completely free of artifacts:

$$PC'_i = \begin{cases} PC'_{i-1} + imm_i & \text{if} \quad bjtaken_i = 1 \quad \wedge \quad I_i\text{is relative branch/jump} \\ RS1_{i-1} & \text{if} \quad bjtaken_i = 1 \quad \wedge \quad I_i\text{is absolute branch/jump} \\ PC'_{i-1} + 4 & \text{otherwise} \end{cases}$$

The delayed program counter is simply computed by

$$DPC_{i+1} = PC'_i.$$

The semantics of the jump and link instructions change by the delayed branch mechanism as well. Saving $PC+4$ into general purpose register $GPR[31]$ results in a return to the delay slot of the jump and link instruction. Of course, the return should be to the instruction *after* the delay slot. Formally, if I_i is a jump and link instruction, then

$$PC_i = PC_{i-1} + 4$$

because I_i is not in a delay slot, and instruction I_{i+1} fetched from address PC_i is the instruction in the delay slot of I_i. The jump and link instruction I_i should therefore save

$$GPR[31]_i = PC_i + 4 = PC_{i-1} + 8.$$

In the simpler delayed PC mechanism, one simply saves

$$GPR[31]_i = PC'_{i-1} + 4.$$

The equivalence of the two mechanisms is asserted in

Theorem 1. *Suppose a machine with delayed branch and a machine with delayed PC are started with identical memory contents and identical contents of the visible registers, then*

1. $(PC_i, PC_{i+1}) = (DPC_i, PC'_i)$,
2. *and if I_i is a jump and link instruction, the value $GPR[31]_i$ saved into register 31 during instruction I_i is identical for both machines.*

Proof. The theorem is proven by induction on i. The case $i = -1$ follows from the rules for initializing PC, $bjtaken$, PC', and DPC. Concluding from $i-1$ to i has two parts. The equation

$$DPC_i = PC'_{i-1} = PC_i$$

follows directly from the definition of DPC and the induction hypothesis. The proof of the equation $PC_{i+1} = PC'_i$ has several cases.

If I_i is a branch or jump, instruction I_i is not in a delay slot, and hence $bjtaken_{i-1} = 0$.

If I_i is a taken branch or a relative jump, it then follows for the target address

$$\begin{array}{lcll} PC'_i & = & PC'_{i-1} + imm_i & \\ & = & PC'_{i-2} + 4 + imm_i & \text{because } bjtaken_{i-1} = 0 \\ & = & PC_{i-1} + 4 + imm_i & \text{by the induction hypothesis for } i-2 \\ & = & btarget_i & \end{array}$$

whereas for an absolute jump it follows

$$PC'_i = RS1_{i-1} = btarget_i.$$

In both cases, $bjtaken_i = 1$; this implies that

$$PC'_i \;=\; btarget_i \;=\; PC_{i+1}.$$

In any other case, $bjtaken_i = 0$ and one concludes

$$\begin{array}{lcll} PC'_i & = & PC'_{i-1} + 4 & \\ & = & PC_i + 4 & \text{by the induction hypothesis} \\ & = & PC_{i+1} & \text{by the definition of delayed branch.} \end{array}$$

For the second part, suppose I_i is a jump and link instruction. With delayed branch, one then saves $PC_{i-1} + 8$. Because I_i is not in a delay slot, it holds

$$\begin{array}{lcll} PC_{i-1} + 8 & = & PC_i + 4 & \\ & = & DPC_i + 4 & \text{by induction hypothesis} \\ & = & PC'_{i-1} + 4 & \text{by definition of delayed PC.} \end{array}$$

This is exactly the value saved in the delayed PC version.

PREPARED SEQUENTIAL MACHINES

The sequential machine DLX_σ is constructed in the following three steps: i) Take a textbook design of a pipelined DLX machine with a classical 5 stage pipeline [14, 13], but without forwarding and interlock. Figure 1 sketches *almost* the data paths of such a machine. Each register, register file or memory is drawn at the end of the stage in which it is written. ii) In stage ID a straightforward circuit $NextPC$ computes the input for PC' which in turn is clocked into DPC (Figure 2). iii) The pipeline stages are updated in a round robin fashion. With proof techniques for sequential machines one shows

Theorem 2. Machine DLX_σ interprets the DLX instruction set with delayed PC semantics.

Theorem 1 implies that machine DLX_σ also interprets the DLX instruction set with delayed branch semantics.

For pipeline stages $k = 0, ..., 4$, nonnegative integers i, and cycles T we denote by

$$I_\sigma(k, T) = i$$

the fact that instruction I_i is in stage k in cycle T. We have

$$I_\sigma(k, T) = i \quad \leftrightarrow \quad T = 5i + k.$$

The content of a register R in cycle T is denoted by R^T.

Suppose execution of instruction I_i is in stage k during cycle T' and the output registers of stage k will be clocked at the end of this cycle. The round robin updating schedule then implies that i) all registers above stage k have already the value they will have after instruction I_i, and ii) all output registers of stages k and below still have the values they had after instruction I_{i-1}. This is asserted in

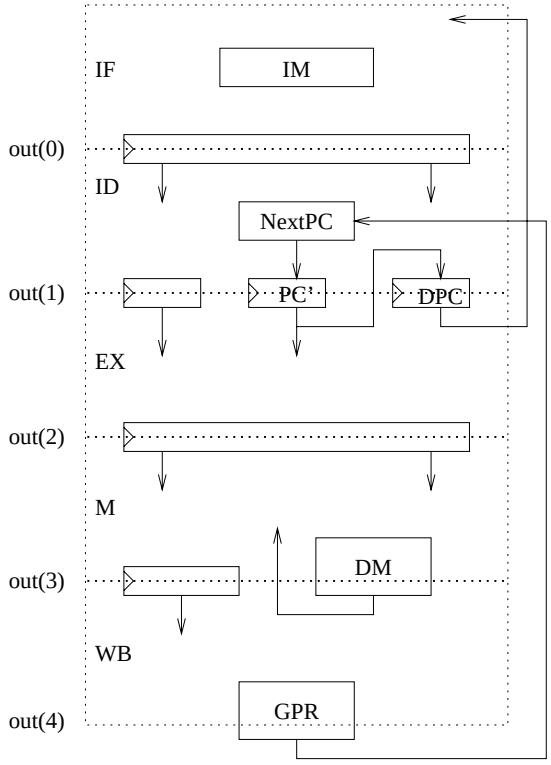

Figure 1 Data flow between the pipeline stages of the DLX_σ design

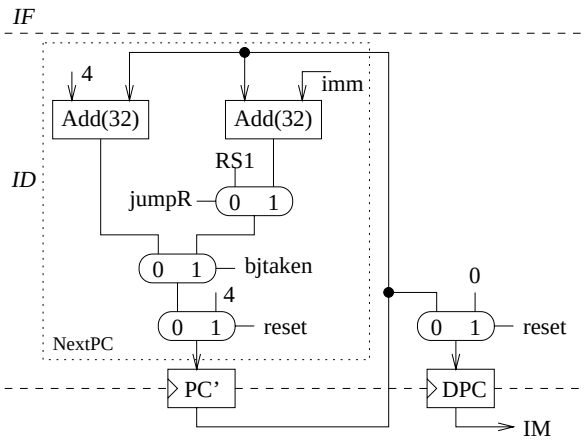

Figure 2 PC environment of the DLX_σ design

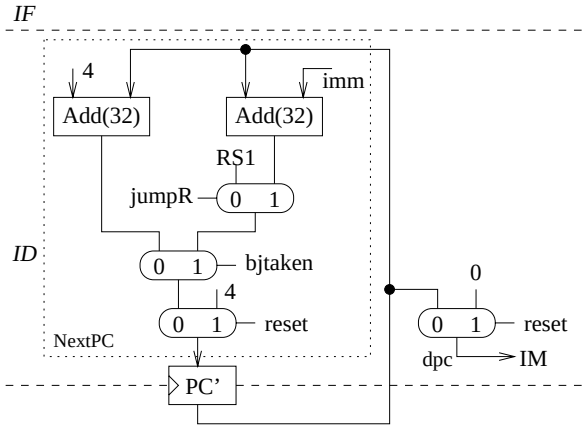

Figure 3 PC environment of the DLX_π design

T	reset	$ue[0]$	$ue[1]$	$ue[2]$	$ue[3]$	$ue[4]$
0	1	1	0	0	0	0
1	0	1	1	0	0	0
2	0	1	1	1	0	0
3	0	1	1	1	1	0
4	0	1	1	1	1	1
...	0	1	1	1	1	1

Table 1 The activation of the update enable signals $ue[4:0]$ after reset. For all i, signal $ue[i]$ enables the update of registers and RAMs in $out(i)$.

Theorem 3. *Let $I_\sigma(k, T') = i$ and let R be an output of stage s. Then*

$$R^{T'} = \begin{cases} R_{i-1} & \text{if} \quad s \geq k \\ R_i & \text{if} \quad s < k \end{cases}$$

A formal proof uses the fact that $R_{i-1} = R^{5i}$ and proceeds for $T = 5i + k$ by induction on k.

PIPELINING AS A TRANSFORMATION

Machine DLX_σ is transformed into a pipelined machine DLX_π in two steps: i) Register DPC is bypassed as shown in Figure 3. This is not surprising; register DPC is an artifact introduced in order to construct a *sequential* machine for a delayed branch semantics. ii) Following reset the stages are updated as indicated in Table 1.

The schedule for this machine is described by the following function I_π:

$$I_\pi(k, T) = i \quad \leftrightarrow \quad T = k + i.$$

stage s	$I_\pi(s,T)$	$I_\pi(s,T-1)$
k-1		i
k	i	i-1

Table 2 Illustration of the scheduling function I_π for the stages $k-1$ and k.

If forwarding and a hardware interlock are added this formula has to be replaced by a more complicated inductive definition [15].

That the pipelined machine simulates the sequential machine is asserted in Theorem 3. In the absence of forwarding and hardware interlock a hypothesis is required about the programs executed.

If we talk about the same register R in the sequential and the pipelined machine, we call one R_σ and the other R_π.

Theorem 4. *Suppose for all i holds that if I_i reads general purpose register R, the instructions I_{i-1}, I_{i-2} and I_{i-3} do not write R. If $I_\pi(k,T) = i$ and R is an output register of stage k, then*

$$R_\pi^{T+1} = R_i.$$

Proof. The proof is done by induction on T. For the cycle $T = 0$, the hypothesis follows from the reset mechanism, e.g.,

$$PC'^1_\pi \;=\; 4 \;=\; PC'_{-1}.$$

The induction step from $T-1$ to T has 5 cases, one for each stage. They all follow the same pattern. Let R be an *input* register of stage k and let T' be the cycle when instruction I_i is in stage k in machine DLX_σ, i.e., $I_\sigma(k,T') = k$. The technical problem is to argue that

$$R_\sigma^{T'} = R_\pi^T.$$

If we can show this for all input registers of stage k then in the corresponding cycles T and T' stage k has in machines DLX_σ and DLX_π the same inputs. Because the stages are identical they produce the same output and the induction step for stage k follows.

The tricky arguments are those dealing with registers R of a stage *below* stage k. We present here only the case $k = 0$ (instruction fetch) and $k = 1$ (decode). For the remaining cases we refer to [15].

Case $k = 0$. In case of stage $k = 0$ (instruction fetch), we have to justify that the delayed PC can be discarded. The input register PC' is an output register of stage 1. We have $I_\pi(0,T) = i$. The scheduling function implies

$$I_\pi(1,T-1) \;=\; I_\pi(0,T-1) - 1 \;=\; i-2.$$

This is illustrated in Table 2. Using Theorem 3 with stage $s = 1$ we conclude

stage s	$I_\pi(s,T)$	$I_\pi(s,T-1)$
1	i	
2	i-1	
3	i-2	
4	i-3	i-4

Table 3 Illustration of the scheduling function I_π for the stages 1 to 4.

$$\begin{array}{rcll} PC'^{T}_{\pi} & = & PC'_{i-2} & \text{by induction hypothesis} \\ & = & DPC_{i-1} & \text{by the construction of delayed PC} \\ & = & DPC^{T'}_{\sigma} & \text{by Theorem 3} \end{array}$$

Case $k = 1$. The induction step for stage $k = 1$ (reading of the operands) uses the hypothesis about the program. In either design, the decode stage has as inputs some registers $R \in out(0)$ and the register file $GPR \in out(4)$.

For $R \in out(0)$, the scheduling function implies

$$I_\pi(0, T-1) \;=\; I_\pi(1, T) \;=\; I_\sigma(1, T') \;=\; i,$$

as illustrated in Table 2. Using Theorem 3 with stage $s = 0$, we conclude

$$R^T_\pi \;=\; R_i \;=\; R^{T'}.$$

If instruction I_i reads a register $GPR[r]$ only the value $GPR[r]^T$ can be used. The scheduling function implies (Table 3)

$$I_\pi(4, T-1) \;=\; i-4.$$

For $i \geq 4$, we conclude using Theorem 3 with stage $s = 4$ that

$$\begin{array}{rcll} GPR[r]^T_\pi & = & GPR[r]_{i-4} & \text{by induction hypothesis} \\ & = & GPR[r]^{T'}_\sigma. & \end{array}$$

According to the hypothesis of the theorem, instructions $I_{i-3}, \ldots, I_{i-1}$ do not write register $GPR[r]$ and hence

$$GPR[r]_{i-1} \;=\; GPR[r]_{i-4}.$$

$i \leq 3$. The update of the register file GPR is enabled by signal $ue[4]$. The stall engine (Table 1) therefore ensures that the register file is not updated during cycles $t \in \{1, 2, 3\}$. Thus,

$$GPR_{-1} \;=\; GPR^1_\pi \;=\; \cdots \;=\; GPR^4_\pi.$$

The hypothesis of the theorem implies that instructions I_j with $0 \leq j < 3$ do not write register $GPR[r]$. Hence,

$$GPR[r]_{-1} \;=\; \cdots \;=\; GPR[r]_{i-1}.$$

By Theorem 3 with stage $s = 4$, we conclude

$$GPR[r]^4_\pi \;=\; GPR[r]_{i-1} \;=\; GPR[r]^{T'}_\sigma.$$

CONCLUSION

Using the construction of delayed PC's we have shown the correctness of a pipelined machine with delayed branch.

References

[1] Phillip J. Windley, "Formal modeling and verification of microprocessors", *IEEE Transactions on Computers*, 1995, 44(1), 54–72.

[2] Mark Bickford and Mandayam Srivas, "Verification of a pipelined microprocessor using Clio", *Proceedings of the Mathematical Sciences Institute Workshop on Hardware Specification, Verification and Synthesis: Mathematical Aspects*, Springer, 1990, volume 408 of *LNCS*, 307–332.

[3] James B. Saxe, Stephen J. Garland, John V. Guttag, and James J. Horning, "Using transformations and verification in circuit design", *Technical report*, Digital Systems Research Center, 1991.

[4] Jerry R. Burch and David L. Dill, "Automatic verification of pipelined microprocessor control, *Proc. International Conference on Computer Aided Verification*, 1994.

[5] Jeremy Levitt and Kunle Olukotun, "A scalable formal verification methodology for pipelined microprocessors", *33rd Design Automation Conference (DAC'96)*, Association for Computing Machinery, 1996, 558–563.

[6] Thomas A. Henzinger, Shaz Qadeer, and Sriram K. Rajamani, "You assume, we guarantee: Methodology and case studies", *Proc. 10th International Conference on Computer-aided Verification (CAV)*, 1998.

[7] W. Damm and A. Pnueli, "Verifying out-of-order executions", *Advances in Hardware Design and Verification: IFIP WG 10.5 Internatinal Conference on Correct Hardware Design and Verification Methods (CHARME)*, Chapmann & Hall., 1997, 23–47.

[8] K.L. McMillan, "Verification of an implementation of Tomasulo's algorithm by composition model checking", *Proc. 10th International Conference on Computer Aided Verification*, 1998, 110–121.

[9] A. Shen and X. Shen, "Using term rewriting systems to design and verify processors", *IEEE Micro Special Issue on Modeling and Validation of Microprocessors*, May/June, 1999.

[10] G. Kane and J. Heinrich, "MIPS RISC Architecture", Prentice Hall, 1992.

[11] SPARC International Inc., "The SPARC Architecture Manual", Prentice Hall, 1992.

[12] Hewlett Packard, "PA-RISC 1.1 Architecture Reference Manual", 1994.

[13] J.L. Hennessy and D.A. Patterson, "Computer Architecture: A Quantitative Approach", *Morgan Kaufmann Publishers*, INC., San Mateo, CA, 2nd edition, 1996.

[14] D.A. Patterson and J.L. Hennessy, "The Hardware/Software Interface", *Morgan Kaufmann Publishers*, INC., San Mateo, CA, 1994.

[15] S. M. Mueller and W. J. Paul., "The Complexity of Simple Computer Architectures II", *Lecture notes*, to appear as a book, 1999. Email: {smueller,wjp}@cs.uni-sb.de.