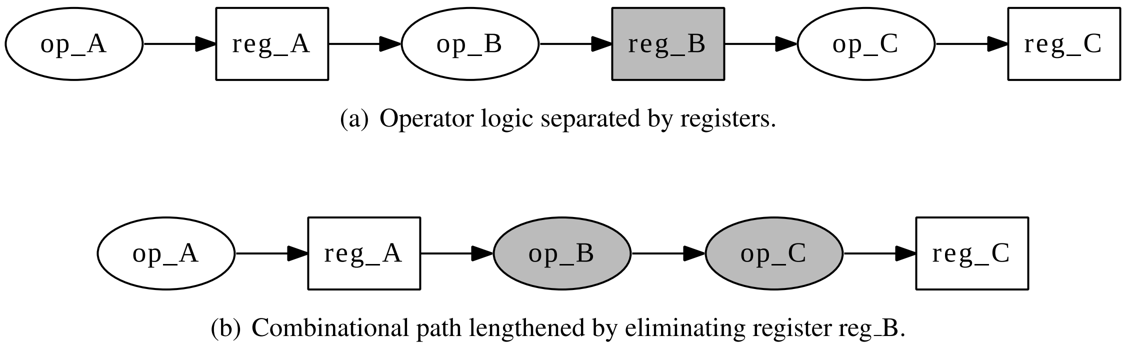
Eliminating registers at the outputs of data-path operators

1 Introduction

Every operator in the data-path has a register at its output, that takes at least one clock cycle before signalling completion. In some cases it may be possible to eliminate this register. The result is a combinational path from the input of the operator to the output register of the next operator. For example, Figure 1(a) shows a sequence of operations ($A \longrightarrow B \longrightarrow C$) where ovals represent combinational logic and boxes represent registers. If the register $\mathrm{reg\_B}$ is removed, the logic for $\mathrm{op\_B}$ and $\mathrm{op\_C}$ is combined into one large combinational path as shown in Figure 1(b).



(a) Operator logic separated by registers.

(b) Combinational path lengthened by eliminating register reg_B.

Figure 1: Eliminating registers at operator outputs.

For this transformation to be correct, we must guarantee that the absence of the register does not result in invalid data at the input of other nodes in the data-path. When $\mathrm{reg\_A}$ is updated, the new value at the output of $\mathrm{op\_B}$ is immediately available at the input of $\mathrm{op\_C}$, in the absence of $\mathrm{reg\_B}$. This may result in an incorrect computation at C, if the actual operation was meant to use the old value stored in $\mathrm{reg\_B}$. In the following sections, we propose a set of necessary and/or sufficient conditions that guarantee that the output of all computations remains valid after eliminating a register.

2 Terminology

Register definition: An "operation"(?) in the control-path that results in updating the value of a register in the data-path is called a *definition* of that register. The set of such definitions of a register A is written as $D_A = \{d_A^0, d_A^1, \ldots\}$.

Register use: A definition d_B^i is said to *use* register A when the value assigned to B is computed in a "non-trivial"(?) manner from the value of A. This is written as $A \mapsto d_B^i$.

Predecessor: A register A is termed as the *predecessor* of a register B if the value of A is used in some definition of B, denoted by the operator "$\longrightarrow$". Also, B is termed as a **successor** of A.

$$A \longrightarrow B \iff \exists d_B^i \text{ such that } A \mapsto d_B^i$$

3 Values latched from external sources

Some operators in the data-path acquire values from the environment, such as memory access operators and input/output ports. The environment is free to change the value after that particular invocation of the operator is done — the register at the output of the operator is required to maintain it until it has been used by other operations in the data-path. Hence that register cannot be eliminated.

Condition 1 *Every definition of the candidate register must assign a value computed from an expression that consists of only constants and values stored in other registers in the data-path.*

4 Registers with conditional definitions

In general, a data-path operator may have multiple control inputs and multiple data outputs (registers). When the operator is invoked by a particular control input, the state machine of the operator determines whether a particular register is defined. Each such definition may assign a different value to the same register. This behaviour results in a multiplexer that selects the result of multiple combinational circuits. If such a register is eliminated, the internal multiplexing (which is dynamic) will drive invalid data at the output.

Condition 2 *Every definition of the candidate register must assign the same value to the register.*

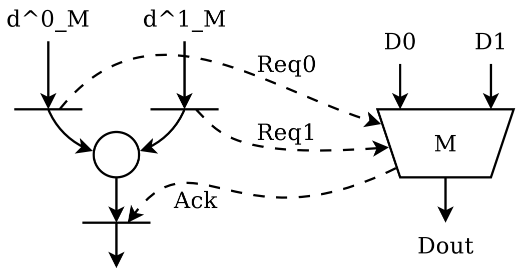


Figure 2: Multiple definitions in a ϕ-function.

For example, the ϕ-function in the AHIR data-path has two control inputs, two data inputs and a single output register, as shown in Figure 2. When definition d_M^0 (or respectively d_M^1) is invoked, the corresponding data-input D_0 (respectively D_1) is assigned to the output register. Eliminating the output register will make it impossible to forward only one of the two data inputs to the output. Hence, the register at the output of a ϕ-function cannot be eliminated.

5 Definition-Use sequences

A register R that satisfies Condition 2 is assigned a value generated by a single combinational circuit. The output of the circuit changes when any of its input registers is updated. In the absence of the register R, this value is immediately visible to the uses of R. Hence R can be eliminated only if the control-path guarantees that when any of the inputs change, R is always updated before it is used.

Condition 3 *For every predecessor P and successor S of the candidate register B, if there is a path from any definition of P to any definition of S that uses the value of B, then that path must pass through some definition of B.*

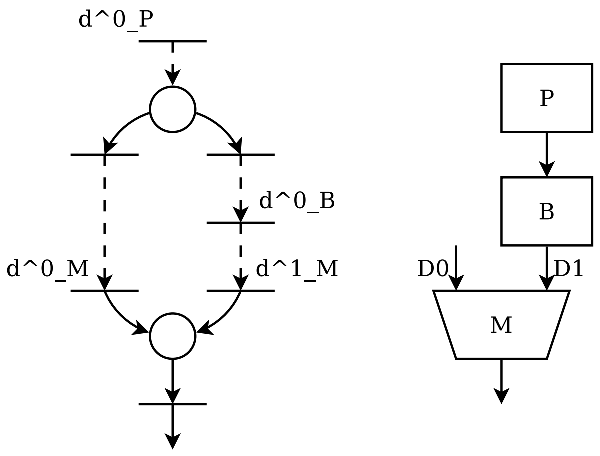


Figure 3: Conditional definitions in the presence of branches.

For example, consider the output of operator B shown in Figure 3. Definition d^0_B is executed only if the appropriate branch is taken in the control-path after definition d^0_P. The value of B is used by definition d^1_M, which occurs along the same path in the control-path. Thus, every path from a definition of P to a definition of M that uses B (there is only one such path) passes through a definition of B. Hence B satisfies this condition for elimination.

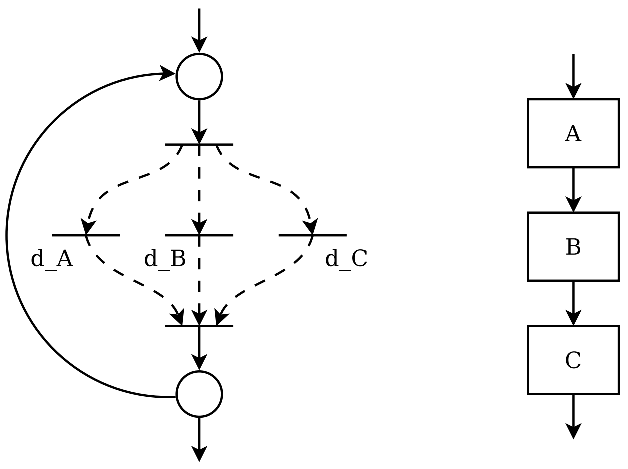


Figure 4: Registers in a pipeline.

Figure 4 shows a pipeline that invokes d_A, d_B and d_C in parallel, where $A \mapsto d_B$ and $B \mapsto d_C$. Register B does not satisfy Condition 3 since there is a path from d_A to d_C which does not pass through d_B. Hence register B cannot be eliminated.

6 Cycles in the data-path

Every cycle in the data-path must have at least one register in order to prevent a combinational loop.

Condition 4 *If the candidate register occurs in a cycle in the data-path, then it must not be the only register in that cycle.*