REAL: A Program for REgister ALlocation

Technical Report CRI-87-10

Fadi J. Kurdahi[1] and Alice C. Parker
Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089-0781

February 9, 1987

[1] Also Supported by the National Council for scientific research, Beirut, Lebanon

Abstract

This report describes the REAL REgister ALlocation program. REAL uses a track assignment algorithm taken from channel routing called the Left Edge algorithm. REAL is optimal for non-pipelined designs with no conditional branches. It is thought that REAL is also optimal for designs with conditional branches, pipelined or not. Experimental results are included in the report, which illustrate the optimal solutions found by REAL. REAL is part of the ADAM Advanced Design AutoMation system, and will be used to process designs output from MAHA and Sehwa.

1. Introduction

One of the important steps in the register-transfer data path synthesis process is register allocation. Frequently, registers are allocated as the operations are scheduled. However, in some synthesis programs which perform global analysis (e.g. MAHA [8]) scheduling of operations into timeslots is not performed a timeslot at a time. Rather, scheduling of an operation into a timeslot may be followed by scheduling of another operation into an earlier or later timeslot. This random scheduling makes concurrent register allocation difficult, since a timeslot may not be completely filled until the last operation remaining in the data flow graph is scheduled.

Therefore, for synthesis programs like MAHA, a register allocation step which occurs after scheduling is more tractable. This paper describes such a scheme. Section 2 describes the specific problem we are solving. Section 3 describes related research. The approach to register allocation we have invented is described in Section 4. Results are presented in Section 5, and conclusions are drawn in Section 6.

2. The Problem Domain

The specific problem REAL (RE*gister* AL*locator*) solves is to input a data flow graph (**dfg**) whose operations have been scheduled, along with a lifetime table of the values in the dfg, and requirements on the number and type of operators used to implement the operations. The scheduling can be either non-overlapped, or overlapped (pipelined) in time. The data flow graph can contain conditional branches.

REAL determines the number of registers required to store values between time steps, and the allocation of values to specific registers. The allocation is guaranteed to be optimal for non-pipelined designs without conditional branches. For the examples processed, optimal results were obtained in all cases, included designs with conditional branches, whether pipelined or not.

3. Related Research

Register allocation has been performed by many synthesis programs, starting with Hafer [3], who attempted to share only temporary registers using heuristics. EMUCS [5] uses a MINIMAX cost criterion, using an algorithm invented by McFarland. Here, scheduling of operations into time steps is done in advance, and register allocation is performed along with operator allocation. Registers and operators are allocated by choosing the values and operations with the lowest implementation costs, and looking ahead to determine the impact on future costs if allocation is postponed. Sharing across conditional branches is supported, but the algorithm has not been implemented on pipelined designs. No discussion of algorithm performance has been published. ELF [2] uses the same method for register allocation.

A formal approach using clique partitioning has been implemented by Tseng [9]. Values become nodes in a graph, and arcs connect values which are never alive during the same timeslot. Clique partitioning has been shown to be NP-Complete, and thus Tseng's formulation of the problem is NP-Complete. However, Tseng implements a polynomial-time solution to the problem which is not guaranteed to be optimal.

4. The Approach to Register Allocation

REAL implements an algorithm for register allocation which has been used previously for track assignment in channel routing. The algorithm is referred to as the Left Edge algorithm and has been proven optimal, and is of complexity $O(n^2)$ [4]. The algorithm has been modified to handle sharing registers between conditional branches, and to handle pipelined designs.

4.1. The Problem Model

A typical instance of the track assignment problem is shown in Fig.4-1, taken from [4]. Essentially, the track assignment problem is solved as follows:

1. sort the wire segments in increasing order of their left edges.

2. Assign the first segment (the leftmost edge) to the first track.

3. Find the first wire whose left edge is to the right of the last selected wire

and assign it to the current track.

4. If no more wires can be assigned to the current track, start a new track and begin again from Step 2. Repeat until all wires are assigned to tracks.

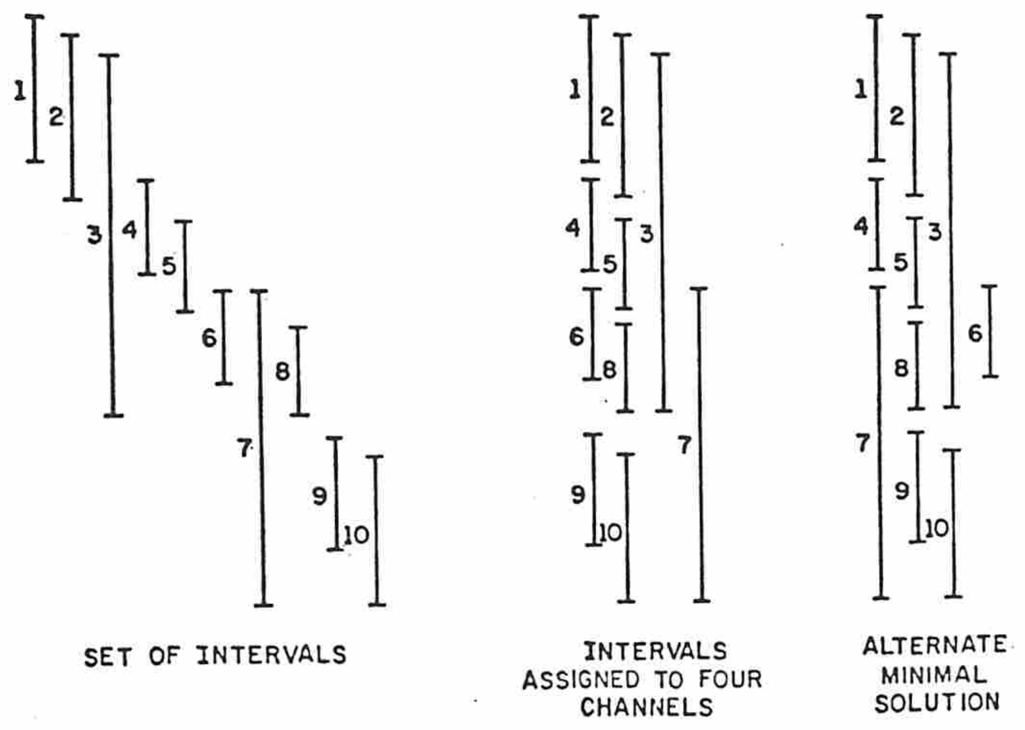
1
2
3
4
5
6
7
8
9
10
SET OF INTERVALS
INTERVALS ASSIGNED TO FOUR CHANNELS
ALTERNATE MINIMAL SOLUTION

Figure 4-1: Track Assignment Example from [4]

The goal of track assignment is to allocate the wire segments to tracks so as to minimize the total number of needed tracks. It is surprising to notice that the left edge algorithm, in spite of taking a greedy approach, does indeed give optimal results.

For dfg's with no loops or conditional branches and non-overlapping scheduling, the register allocation problem is identical to the track assignment problem as described above. In our problem, we model registers as tracks, and values as wires. The left and right edges of wires correspond to the birth and death times of values, respectively.

Birth and death times of a value are defined as the time of definition[1] and the last time step the value is used. The time span between birth and death is referred to as the lifetime of a value. The set of values and their lifetimes is called the lifetime table. An example dfg and its corresponding lifetime table are shown in Fig. 4-2 and Table 4-1, respectively. This terminology has its roots in the theory of compiler design [1].

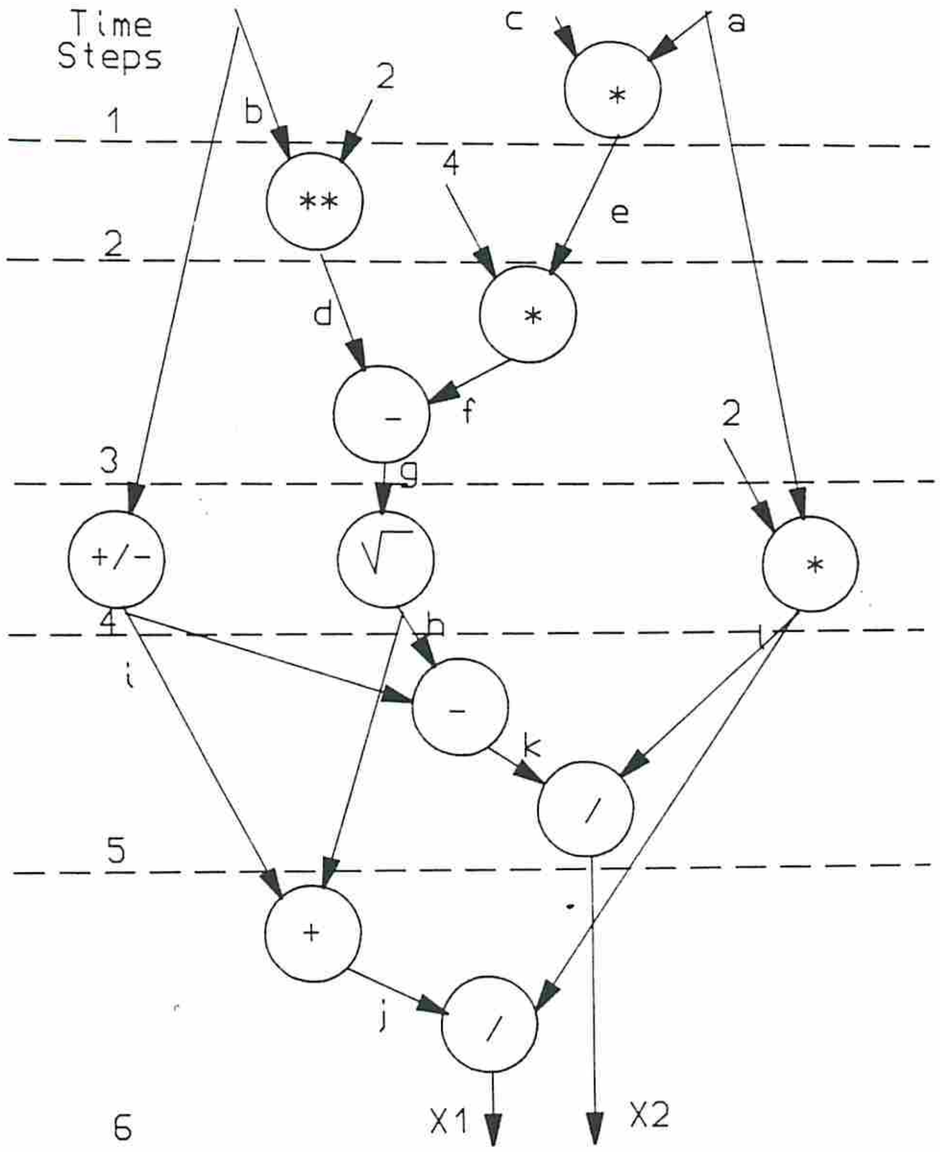

Figure 4-2: An Example Data Flow Graph (DFG)

Given a lifetime table for a dfg, the goal is to assign values(wires) to registers(tracks) so

[1]time of definition is the starting time step for input values, or the first time a value appears in the left hand side of an evaluation expression.

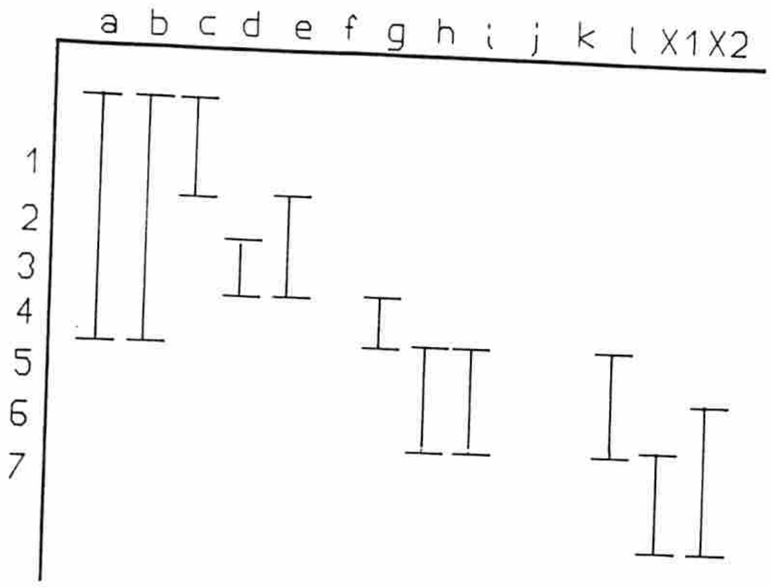

Table 4-1: An Example Lifetime Table for Fig. 4-2

as to minimize the total number of registers(tracks) needed to store the values. Two values cannot share a register if they overlap in time, whereas two wire segments cannot share a track if they overlap in space. When applied to our problem, the Left Edge algorithm allocates values to registers in a way which minimizes the number of registers needed for storage. In Fig. 4-2, we show an example optimal allocation of the values in Table 4-1 to registers.

```
Number of values = 11
Number of time steps = 8
Number of registers needed = 4
Allocation of values to registers
Register #: value1, value2,...

Register 1 : a ,h ,X1
Register 2 : b ,i
Register 3 : c ,e ,g ,l
Register 4 : d ,X2
```

Table 4-2: Register Allocation for Table 4-1

4.2. Handling Conditional Branches

In order to handle conditional branches, we must color the arcs of the graph to indicate mutual exclusion of values. The algorithm used for the coloring, called the MPC algorithm, is a modified version of the algorithm used by Park for determining mutual exclusion of operations [6] [7]. Two values are considered mutually exclusive if they can never be alive at the same time. Conditional branches are represented in our model by Distribute-Join (DJ) blocks. An example of such a block is shown in Fig. 4-3.

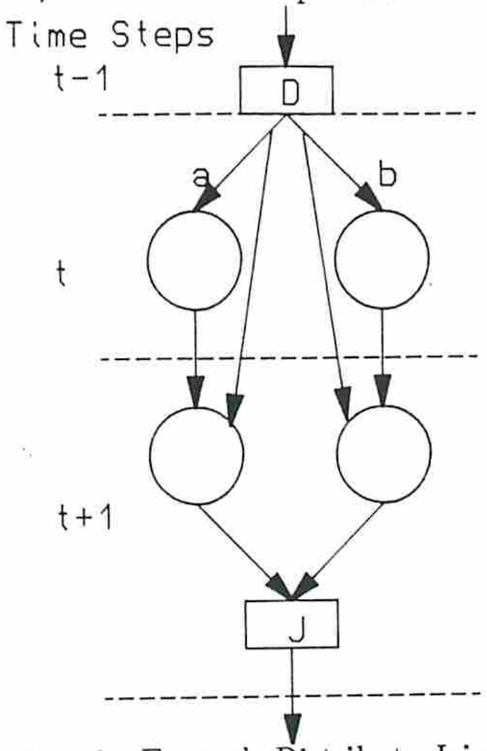

Figure 4-3: An Example Distribute-Join (DJ) Block

Only one of the branches will be traversed during an execution cycle, depending on the condition at the entry of the DJ block. Values *a* and *b* belong to different branches, hence they are mutually exclusive and can share a register for storage (even though their lifetimes are overlapping) since they can never exist at the same time.

The MPC algorithm assigns a specific color to each value. using these colors, the allocation program can check whether two values are mutually exclusive and thereby whether or not they can share a register.

Fig. 4-4 shows a dataflow example with conditional branches; the value colors are also shown. Table 4-3 shows the lifetime table. Table 4-4 shows the final allocation of values to registers. While this allocation is not guaranteed to be optimal, all the examples we ran so far have produced optimal allocations. This is an indication that the algorithm may still be optimal in the existence of conditional branches, or at worst, would produce near optimal solutions.

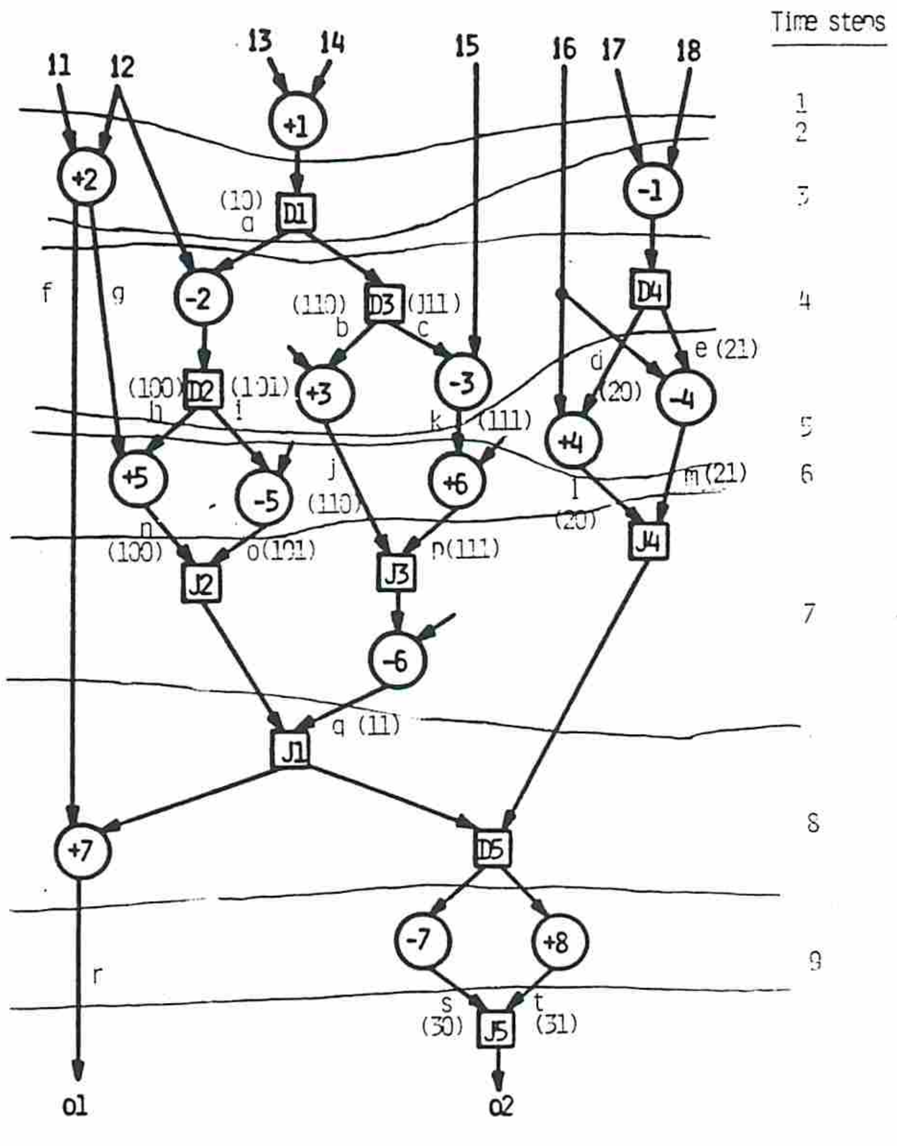

Figure 4-4: A Data Flow Graph with Conditional Branches

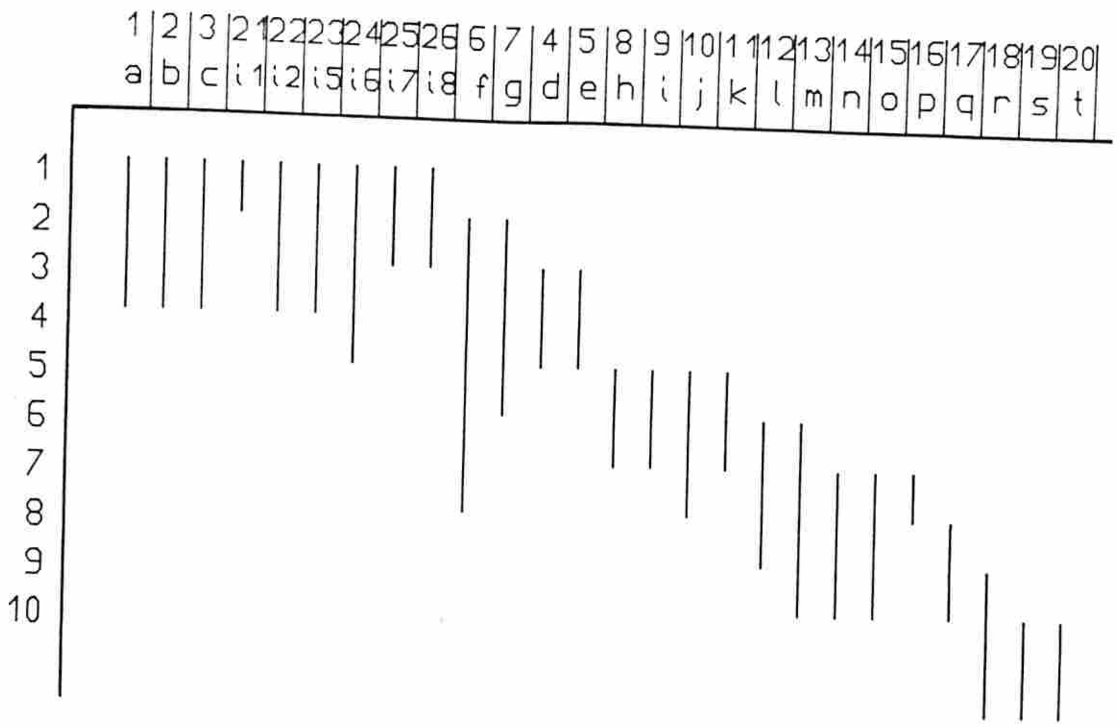

Table 4-3: Lifetime Table for DFG in Fig. 4-4

```
Number of values = 26
Number of time steps = 10
Number of registers needed = 8
Allocation of values to registers
Register #: value1, value2,...

Register 1 : 1 ,2 ,3 ,8 ,9 ,10 ,11 ,14 ,15 ,16 ,17 ,19 ,20
Register 2 : 21 ,6 ,18
Register 3 : 22 ,12 ,13
Register 4 : 23
Register 5 : 24
Register 6 : 25 ,4 ,5
Register 7 : 26
Register 8 : 7
```

Table 4-4: Register Allocation for the Table in 4-3

4.3. Extensions to Pipelined Data Paths

The register allocation technique described above was extended to handle pipelined data paths. When a graph schedule is pipelined, timesteps are overlapped. The graph shown in Fig. 4-4 is now scheduled with a latency L=3. This means, for example, that timesteps 1,4 and 7 will be executing concurrently on three consecutive sets of values. To account for the multiple instances of values, the lifetime table must be extended to represent *streams* of values instead of a single occurrence of each value. These streams will occur with a period of L timesteps, i.e. a new instance of a particular value will be generated every L timesteps. It is interesting to note that if a value is alive for more than L timesteps, then more than one instance of that value may exist at the same time and will have to be stored in different registers.

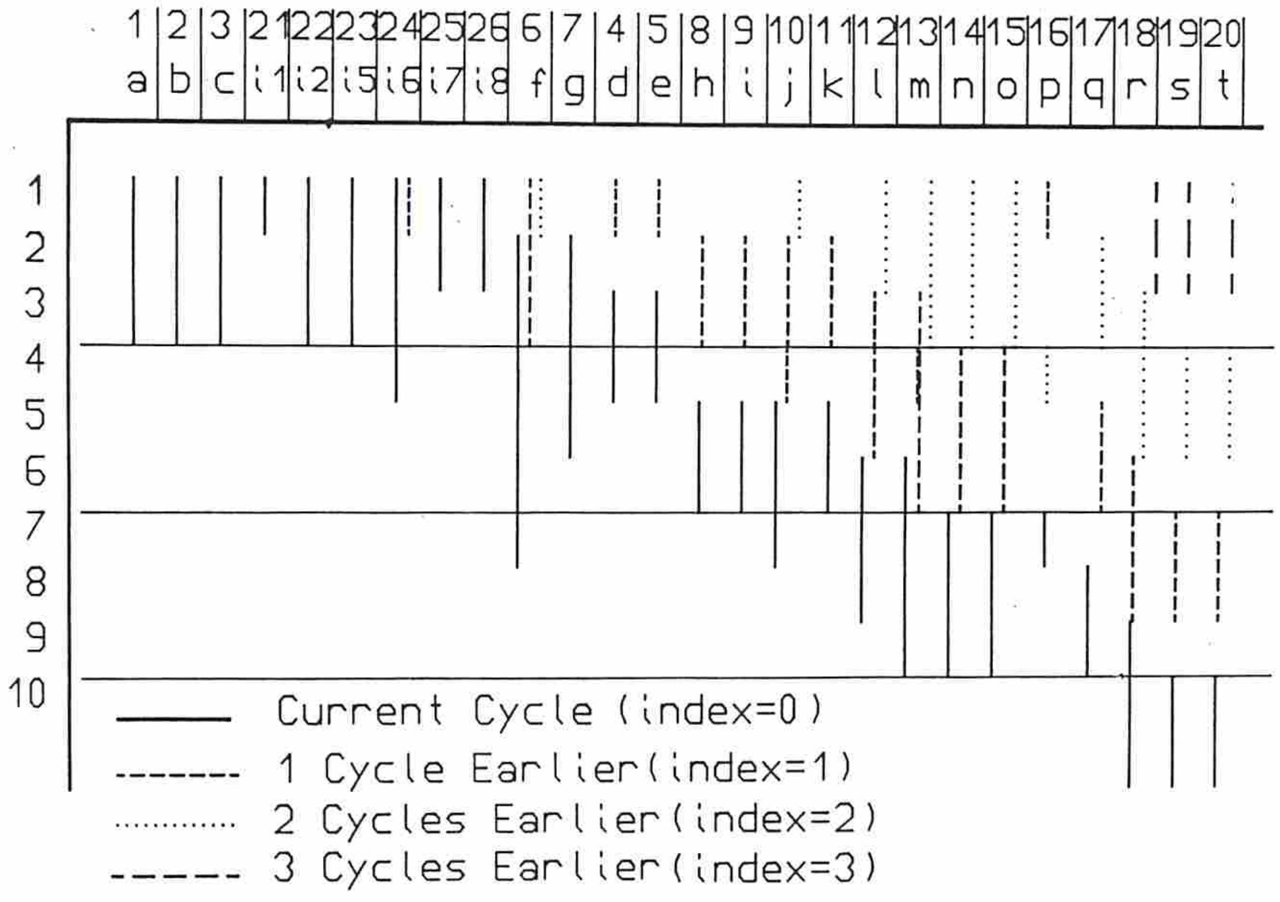

Table 4-5: Lifetime Table for Pipelined Scheduling

Since the whole pipeline cycle is repeated every L timesteps, we only need to consider a *window* of L timesteps to allocate values to registers. This means that the pipeline lifetime table need only be L timesteps long. Table 4-5 shows the new lifetime table derived from the pipelined scheduling of the example in Fig. 4-4 with latency L = 3. Here, we show several latency windows in order to better visualize the streams of values.

Conditional branches in the pipelined schedules require more attention. Two overlapping and mutually exclusive values can share a register *iff* they belong to the *same* data set, i.e. they must be in the *same* time step, not in overlapping time steps. This is due to the fact that the branching conditions may vary from one data set to another. Fig. 4-5 depicts such a situation. Values x and y are mutually exclusive, since they belong to different branches. If the latency L is 2, then timesteps 2 and 4 will be overlapping, and x and y must be stored at the same time. But, these two values cannot share a register because they belong to different data sets. This is so because the branching condition during the cycle when x was generated may be different than the one during the next cycle, when y was generated. This means that x and y can indeed exist at the same time and must be stored in separate registers.

To account for this peculiarity, each value in the lifetime table is assigned a time index indicating which cycle it belongs to. Overlapping values with different time indexes cannot share registers even if their colors indicate that they are mutually exclusive.

5. Current Status and Results

REAL has been implemented in C and runs under UNIX 4.2 bsd on SUN 3 workstations. Fig. 5-1 shows the register allocation generated by REAL for the example in Fig. 4-4 and Table 4-5. The resulting allocation took less than a second of cpu time for 36 values.

REAL is intended as a postprocessor for the current synthesis programs being used at USC, MAHA and Sehwa, which generate non-overlapped and pipelined designs, respectively. We are currently interfacing REAL with these two programs as part of an completely integrated synthesis tool.

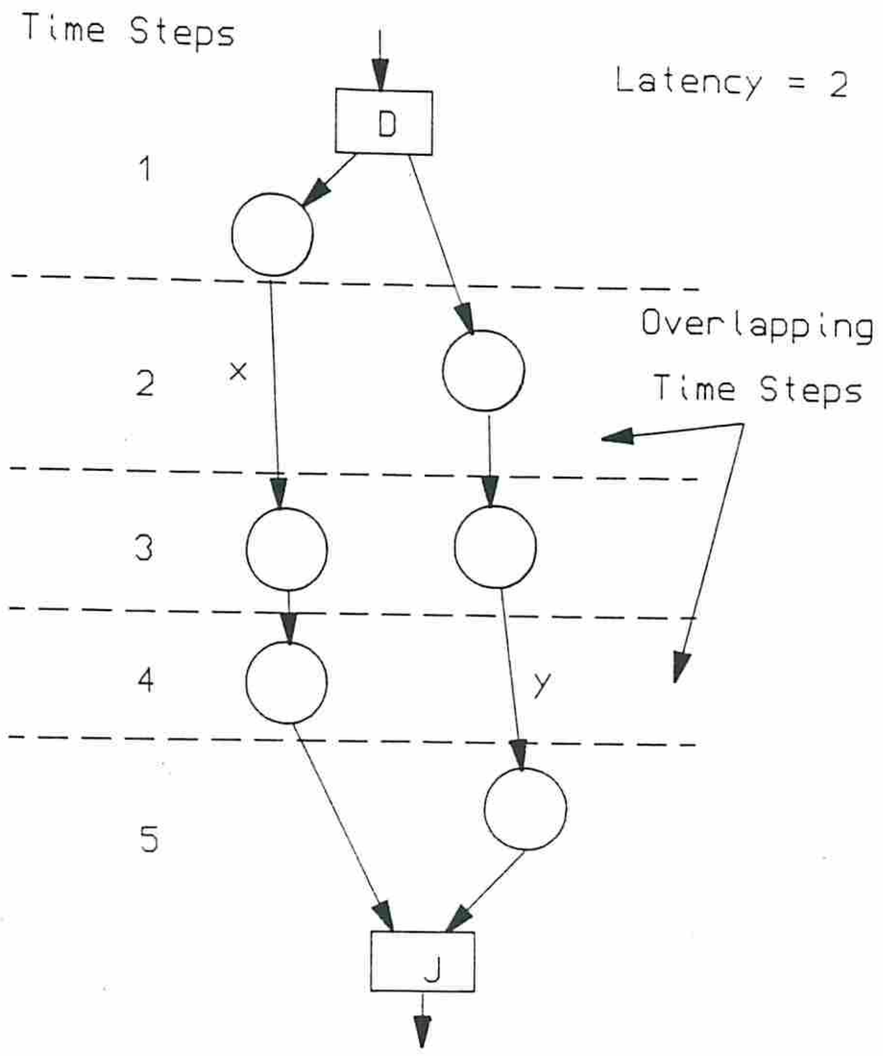

Figure 4-5: Conditional Branches in Pipelined Scheduling

6. Conclusions and Future Research

This paper has described REAL, a program for register allocation. REAL has found optimal solutions to all examples tested so far. Since the complexity of the algorithm is low, REAL can be used in a practical design environment. REAL is useful in ADAM, since it forms the next step in the design process after Sehwa and MAHA have been run.

Since Sehwa and MAHA do not perform actual allocation of operators, but only produce a feasible allocation scheme, the register allocation proceeds without optimizing

```
Number of values = 36
Number of time steps = 4
Number of registers needed = 17
Allocation of values to registers
Register #: value1:index1, value2:index2,...

Register 1 : 1:0, 2:0, 3:0,
Register 2 : 4:1, 5:1, 6:0,
Register 3 : 6:2, 7:0,
Register 4 : 6:1,
Register 5 : 7:1, 4:0, 5:0,
Register 6 : 10:2, 14:2, 15:2, 17:2,
Register 7 : 12:2, 13:2,
Register 8 : 16:3, 8:1, 9:1, 10:1, 11:1,
Register 9 : 18:3, 12:1, 13:1,
Register 10 : 19:3, 20:3, 18:2,
Register 11 : 21:0,
Register 12 : 22:0,
Register 13 : 23:0,
Register 14 : 24:1,
Register 15 : 24:0,
Register 16 : 25:0,
Register 17 : 26:0,
```

Figure 5-1: REAL's Allocation for the Example in Fig. **4-4**

the multiplexing cost. However, operator allocation, which will be performed afterwards, will have to take this into account.

References

[1] Aho, A. and Ullman J.
Principles of Compiler Design.
Addison-Wesley, Reading, MA, 1977.

[2] Girczyc, E. and Knight, J.
An ADA to Standard Cell Hardware Compiler Based on Graph Grammars and Scheduling.
In *Proceedings, 1984 International Conference on Computer Design - ICCD*, pages 726-729. October, 1984.

[3] Hafer,L., Parker,A.
Register-Transfer Level Digital Design Automation: The Allocation Process.
In *Design Automation Conference Proceedings no. 15*, pages 213-219. ACM SIGDA, IEEE Comp. Soc. Tech. Com. on Design Automation, June, 1978.

[4] Hashimoto, A. and Stevens, J.
Wire routing by optimizing channel assignment within large apertures.
In *Proceedings 8th DA workshop*, pages 155-169. IEEE, 1971.

[5] Hitchcock, C.Y.
Automated Synthesis of Data Paths.
Master's thesis, Carnegie-Mellon University, 1983.

[6] Park, N.
Synthesis of High-Speed Digital Systems.
PhD thesis, Dept. of Electrical Engineering, University of Southern California, September, 1985.

[7] Park, N., and Parker, A.C.
Sehwa:A Program for Synthesis of Pipelines.
In *Proceedings of the 23rd Design Automation Conference*, pages 454-460. IEEE and ACM, July, 1986.

[8] Parker, A.C., Pizarro, J. and Mlinar, M.
MAHA: A Program for Datapath Synthesis.
In *Proc. 23rd Design Automation Conf.*, pages 461-466. IEEE and ACM, June, 1986.

[9] Tseng, C.-J., and Siewiorek, D.P.
Automated Synthesis of Data Paths in Digital Systems.
IEEE Trans, on CAD CAD-5(3):379-395, July, 1986.