RESOURCE ALLOCATION AND ASSIGNMENT

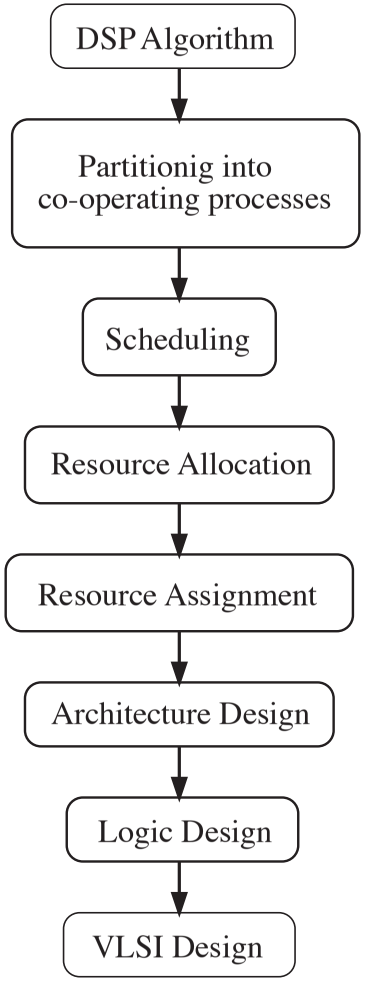
In the resource allocation step the amount of resources required to execute the different types of processes is determined.

We will refer to the time interval during which a process is executed as the *lifetime* of the process.

Note that storing a value in memory is a (storage) process.

The amount of resources can be minimized by letting processes that do not overlap in time share resources.

The number and type of resources can be determined from the operation schedule and corresponding lifetime table for the signal values.



Usually, the largest number of concurrent processes determines the number of resources required.

In fact, both the resource allocation and assignment steps are usually simple one-to-one mappings where each process can be bounded to a free resource.

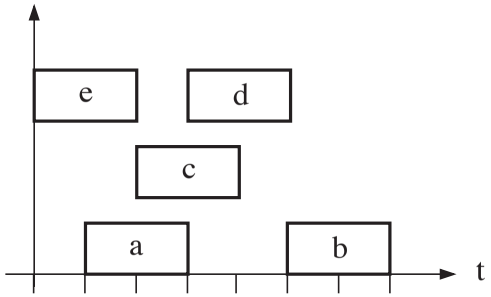
However, as shown in Example 7.8, more resources than there are concurrent processes are required in some cases.

This fact makes it difficult to evaluate the cost function to be minimized by the scheduling.

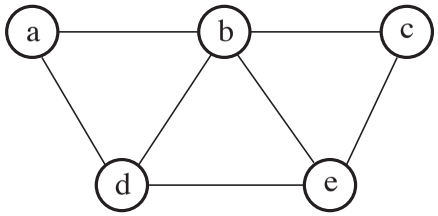
Clique Partitioning

The resource allocation problem can be solved by using different types of graphs to determine whether two processes of the same type may, or may not, share a resource.

The *connectivity graph* which is also called a *compatibility graph* is obtained by connecting two vertices with an edge if the lifetimes of the corresponding processes do not overlap. This implies that the processes may share a resource.



Schedule with five operations (processes)



Connectivity graph

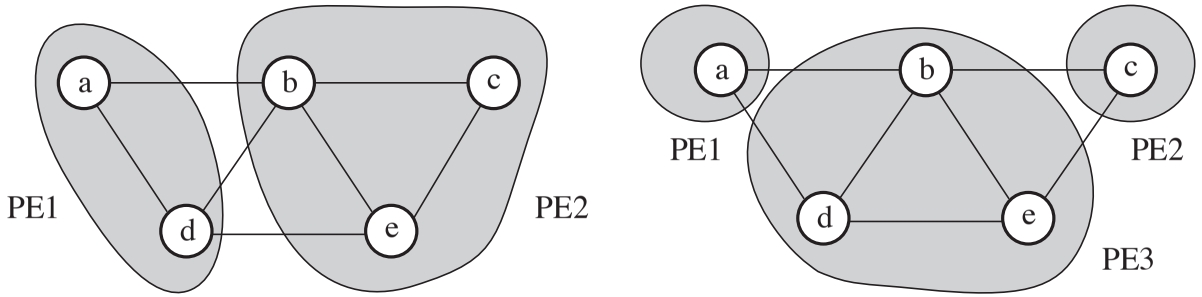
In order to determine which processes can share resources, we partition the connectivity graph into a number of *cliques*.

A *clique* is defined as a fully connected subgraph that has an edge between all pairs of vertices.

Thus, the processes corresponding to the vertices in a clique may share the same resource.

Now, we choose the cliques so that they completely cover the connectivity graph—i.e., so that every vertex belongs to one, and only one, clique.

Hence, each clique will correspond to a set of processes that may share a resource and the number of cliques used to cover the connectivity graph determines the number of resources.



RESOURCE ASSIGNMENT

The resource assignment step involves selection of a particular resource from a pool of resources to execute a certain process.

Ideally this assignment should be made at the same time as the scheduling, but this is not done in practice. Because of the complexity of the problem, it is instead divided into separate scheduling, allocation, and assignment steps.

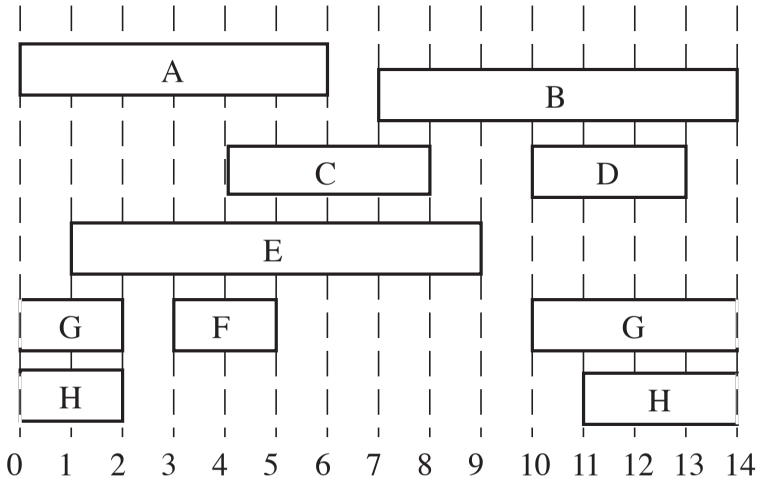
The problem of assigning a class of processes to a corresponding set of resources can be formulated as a scheduling problem where processes with compatible lifetimes are assigned to the same resource.

The Left-Edge Algorithm

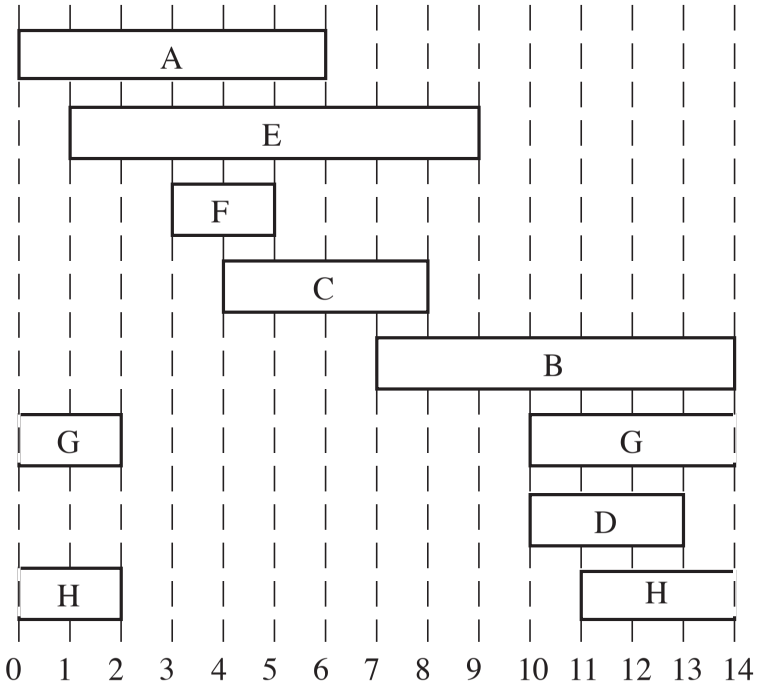
The left-edge algorithm is a heuristic list-scheduling method.

```
1.  Sort the processes into a list according to their starting times.
        Begin the list with the process having the longest lifetime if
        the schedule is cyclic.
        Processes with the same starting time are sorted with the
        longest lifetime first. Let i = 1.

2.  Assign the first process in the list to a free resource i, determine
    its finishing time, and remove it from the list.

3.  Search the list for the first process that has

    a)  a starting time equal to, or later than, the finishing time
        for the previously assigned process; and
    b)  a finishing time that is no later than the starting time for
        the first process selected in step 2.

4.  If the search in step 3 fails then
        if there are processes left in the list then
            let i ← i + 1 and repeat from step 2
        else
            Stop
        endif;
        assign the process to resource i, determine its finishing
        time, remove it from the list, and
        repeat from step 3.
    endif.
```

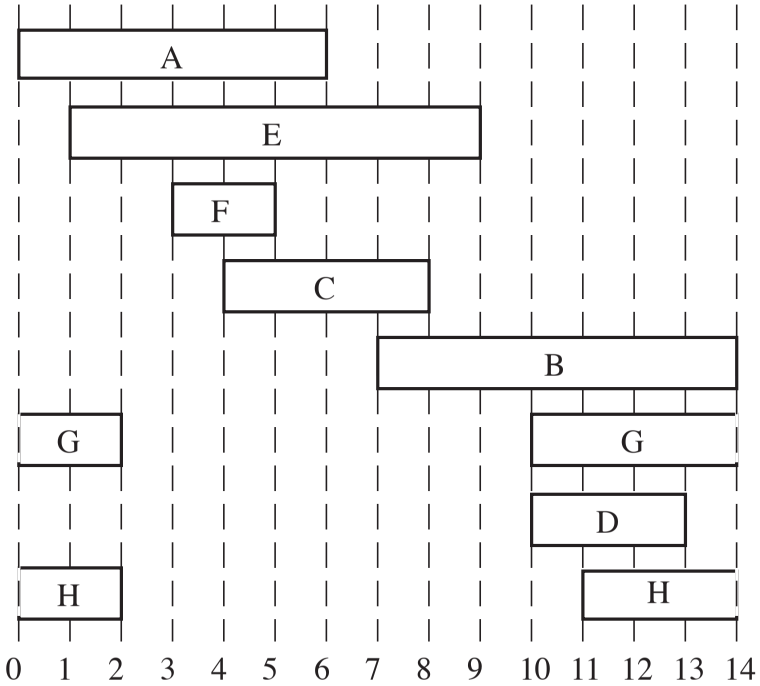
Example 7.10



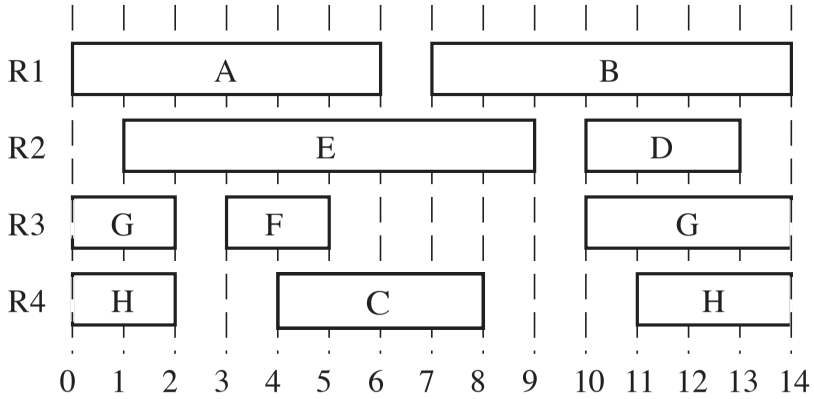
Processes



Sorted processes



Sorted processes



Resource assignment

* No “gain” in this case!

INTERPOLATOR, CONT.

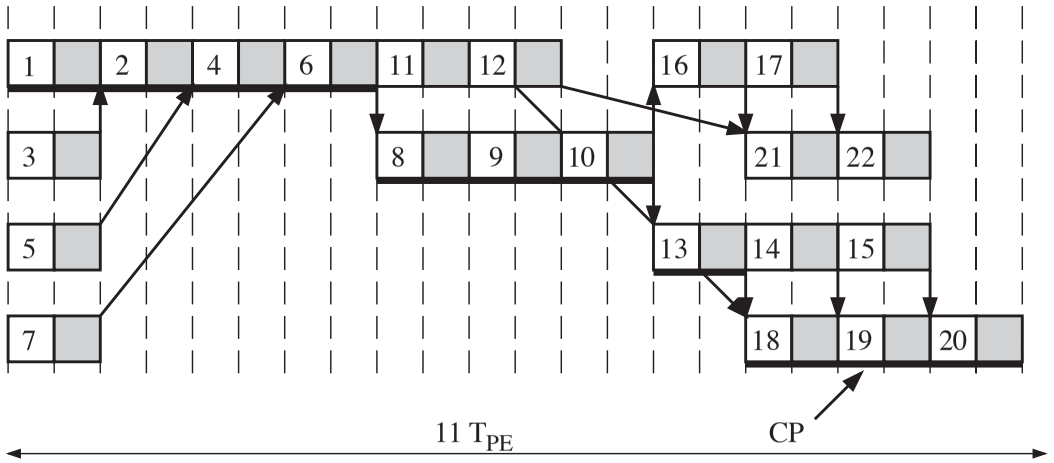
We will determine a suitable operation schedule, and perform resource allocation and assignment, for the interpolator.

We assume that the PEs have two pipeline stages.

This means that the outputs of an adaptor can not be used directly as inputs to an adjacent adaptor, since the signal values are inside the pipeline.

An operation can be executed in each time slot, but the result will be first available after two time slots.

Using the precedence relations we get the ASAP schedule.



In this case it is enough to perform the scheduling over only one sample interval, since the critical loops contain only one delay element.

The execution time for a bit-serial PE is $W_d + 3$ clock cycles, where W_d is the data word length.

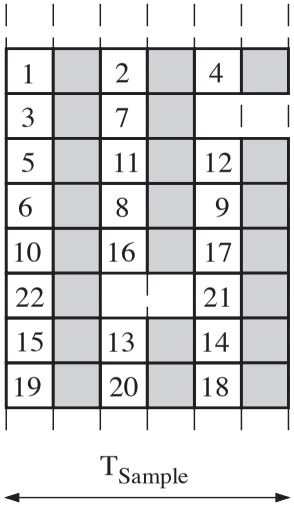
The latency is $2(W_d + 3) = 2$ time units, because of the pipelining.

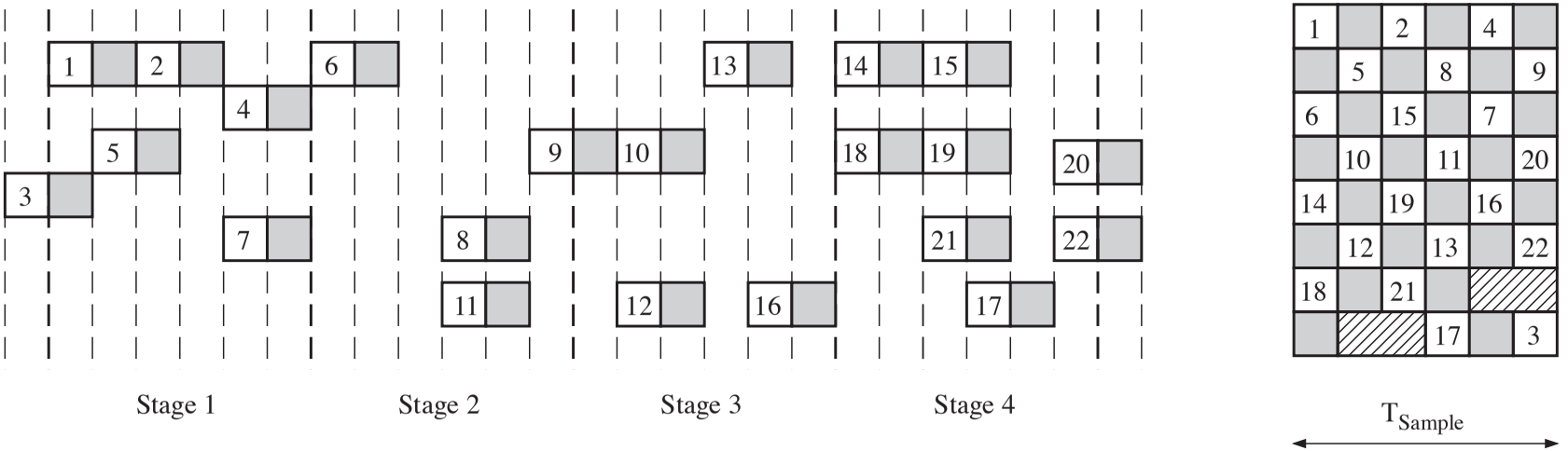
We estimate that $W_d = 21$ bits are required and that the minimum clock frequency for the PEs is at least 220 MHz.

The minimum number of PEs is

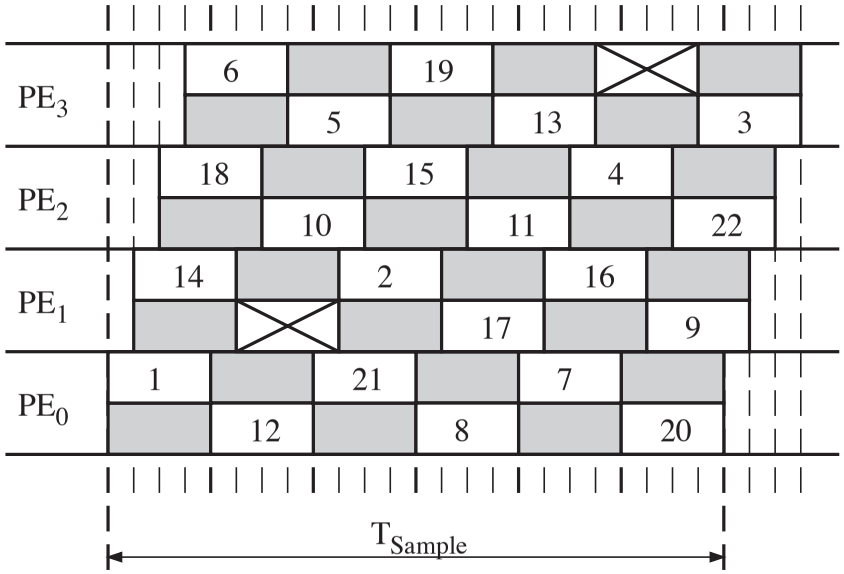
$$N_a = \frac{(W_d + 3) N_{op} f_{sample}}{f_{CL}} = \frac{24 \cdot 22 \cdot 1.6 \quad 10^6}{220 \quad 10^6} \approx 3.84$$

We choose to use four PEs to perform the 22 adaptor operations. Hence, $T_{sample} = 6$ TPE.


1 2 4
3 7
5 11 12
6 8 9
10 16 17
22 21
15 13 14
19 20 18
T Sample


Stage 1
Stage 2
Stage 3
Stage 4
T Sample

Final Processor Assignment

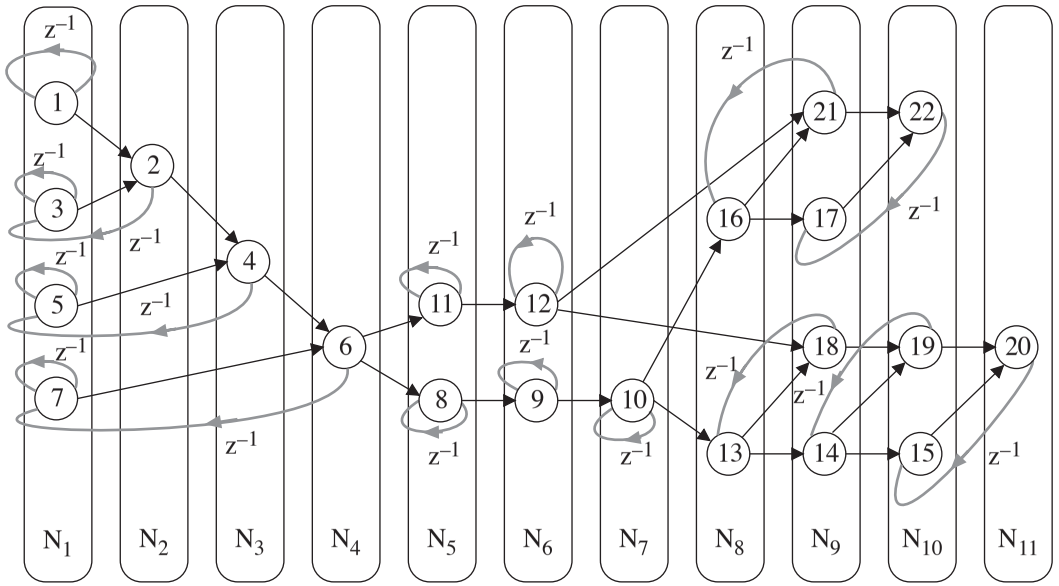


Skewed execution of the PEs

Memory Assignment

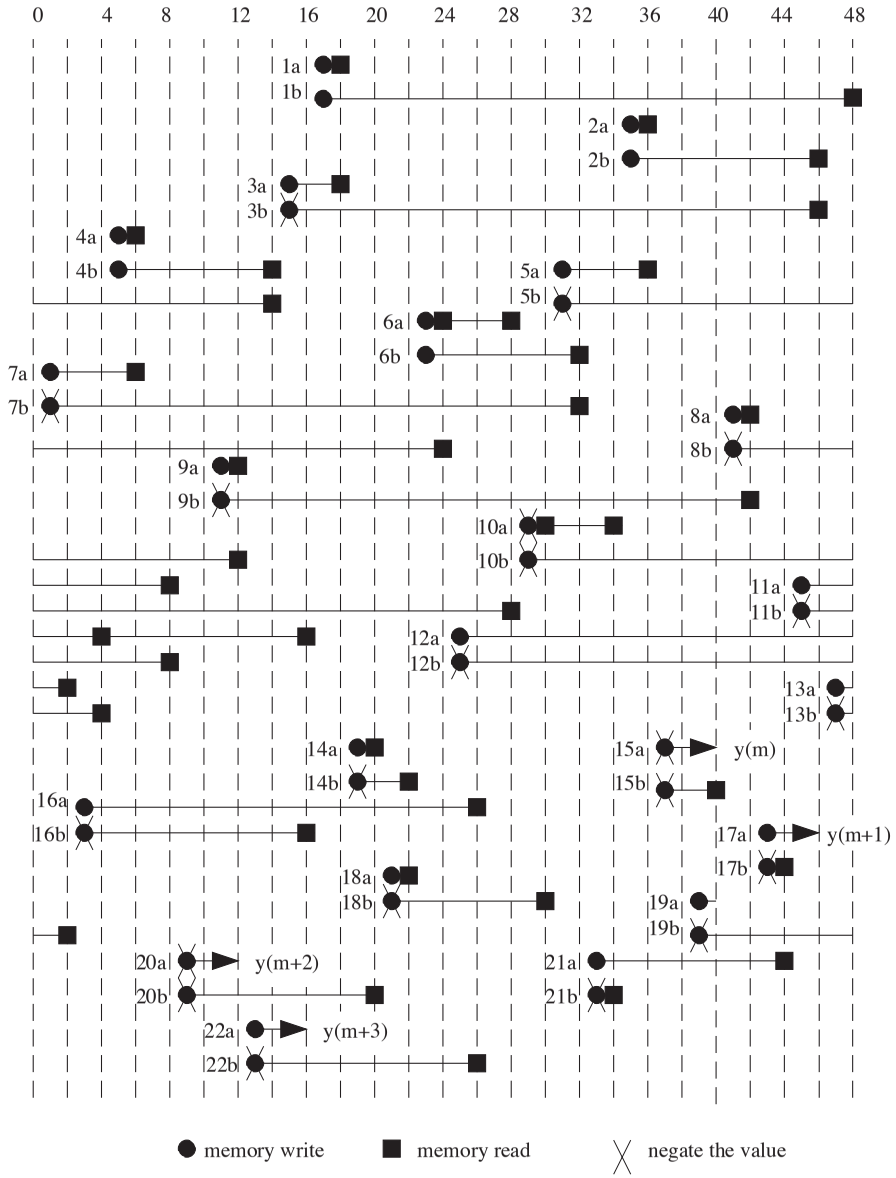
The constraints on memory allocation and assignment can be derived from the PE schedule.

The memory–PE transactions can be extracted from the PE schedule and the precedence form.



There are four transactions (two inputs and outputs, each requiring one read and one write operation) taking place in each T_{PE} time slot.

The memory schedule will therefore be over $2 \cdot 4 \cdot 6 = 48T_M$ time slots, where T_M is the time required for a memory read or a write operation.


0
4
8
12
16
20
24
28
32
36
40
44
48
1a
1b
2a
2b
3a
3b
4a
4b
5a
5b
6a
6b
7a
7b
8a
8b
9a
9b
10a
10b
11a
11b
12a
12b
13a
13b
14a
14b
15a
15b
y(m)
16a
16b
17a
17b
y(m+1)
18a
18b
19a
19b
20a
20b
y(m+2)
21a
21b
22a
22b
y(m+3)
memory write
memory read
negate the value

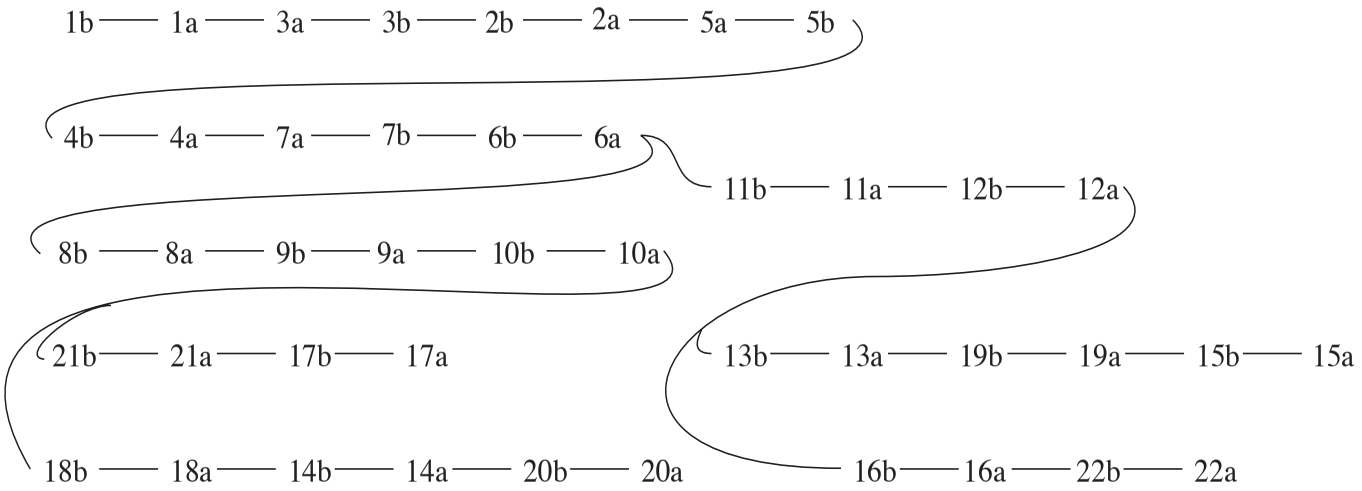
We have chosen to use two memories, since the adaptors have two inputs and two outputs.

We will use a modified version of the clique partitioning technique described above to assign the variables to the memories.

Instead of drawing a line between processes that may share resources, we will draw a line between vertices (memory variables) that can not be stored in the same memory or are accessed at the same time instances.

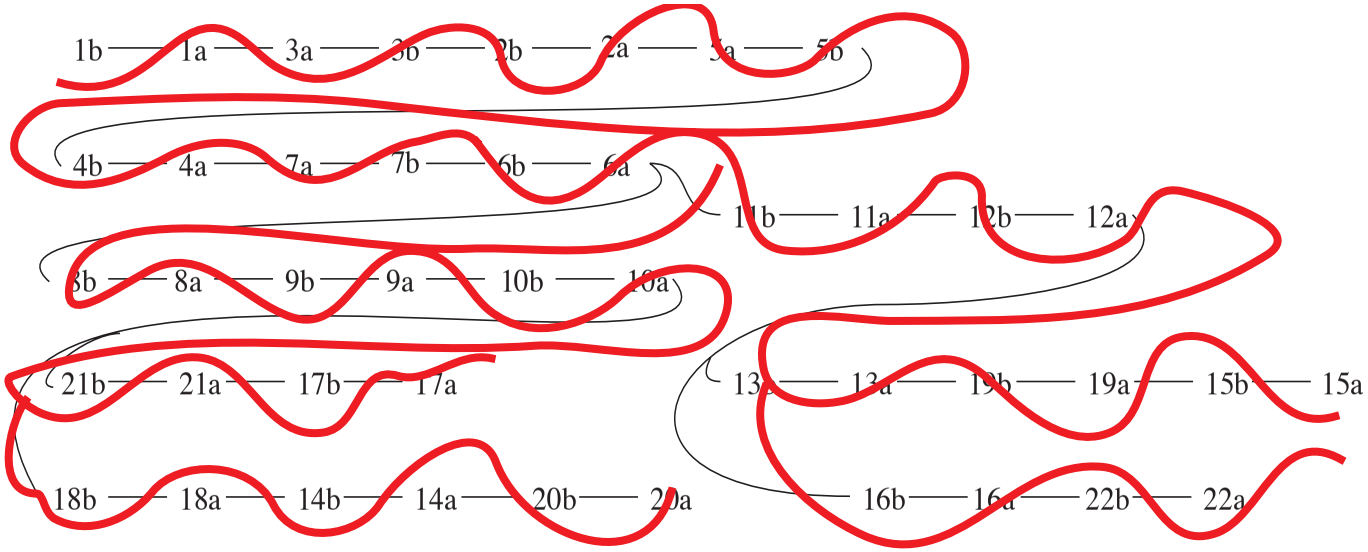
The reason for this is that the number of branches in the graph otherwise becomes unmanageably large.

The resulting graph is called an exclusion graph.



This graph must now be partitioned into two cliques (two memories) by drawing a line that separates all connected vertices.

This procedure separates variables that must be stored in different memories.


1b
1a
3a
3b
2b
2a
5a
5b
4b
4a
7a
7b
6b
6a
11b
11a
12b
12a
8b
8a
9b
9a
10b
10a
21b
21a
17b
17a
13b
13a
19b
19a
15b
15a
18b
18a
14b
14a
20b
20a
16b
16a
22b
22a

Memory Cell Allocation and Assignment