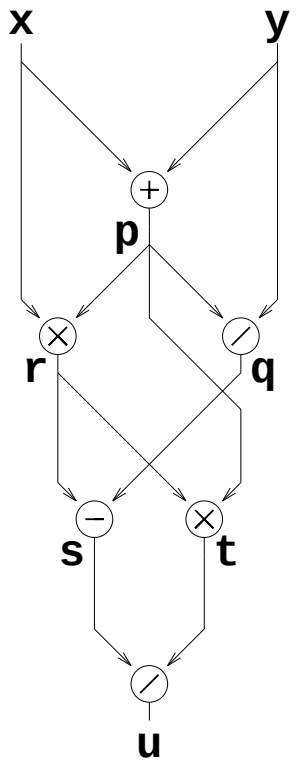
Dataflow Computers

Control flow and data flow representations of a program:

```
p := x+y;
q := p/y;
r := x*p;
s := r-q;
t := r*p;
u := s/t;
```

Single Thread
Control Flow



Data Flow Graph

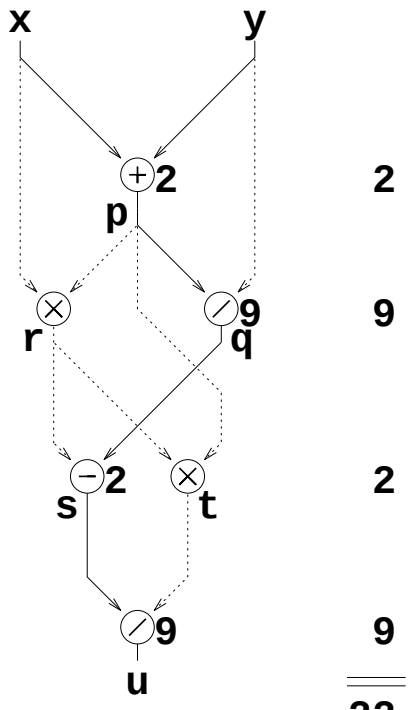
```
SEQ
  p := x+y
  PAR
    q := p/y
    r := x*p
  PAR
    s := r-q
    t := r*p
  u := s/t
```

Multiple Thread
Control Flow

Dataflow Computers

Consider the critical path:

```
SEQ
  p := x+y          2   2
  PAR
    q := p/y        9 } 9
    r := x*p        4 }
  PAR
    s := r-q        2 } 4
    t := r*p        4 }
  u := s/t          9   9
                        ==
                       24
```

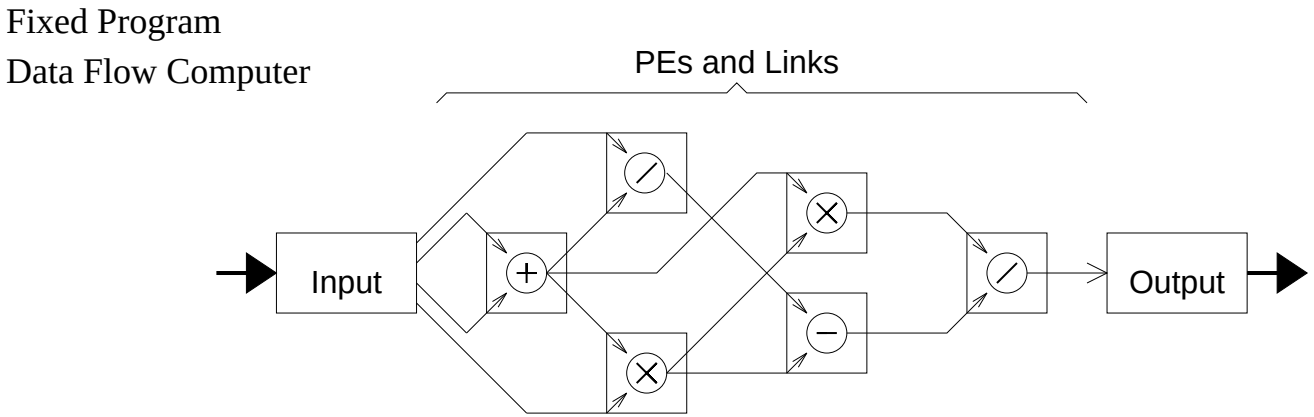



```
+   2 cycles
-   2 cycles
*   4 cycles
/   9 cycles
```

Our simple control flow description does not exploit the early availability of `r` in order to take `t` out of the critical path. If we produce a dataflow machine based on execution of the dataflow graph we can exploit the full parallelism with ease.

Fixed Program Dataflow Computers[1]

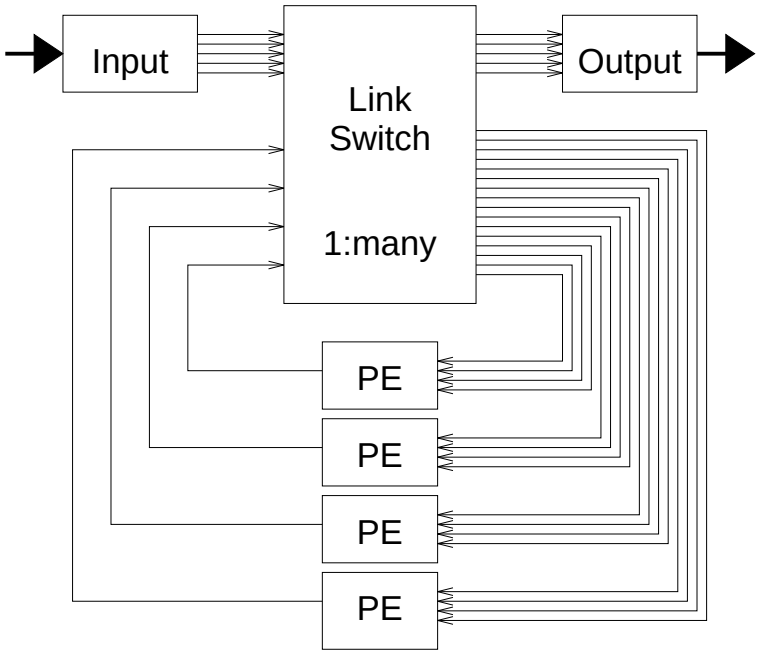
- Each *node* on the dataflow graph maps to a PE (*Processing Element*) in the architecture.
- Each *arc* on the dataflow graph maps to a *link* in the architecture.
- The node is said to *fire* when it receives sufficient *data tokens* on its input links.
- Computation is driven by the flow of data tokens between the nodes.



[1]These machines are often used in signal processing where a constant stream of data values is to be processed.

Programmable Dataflow Computer

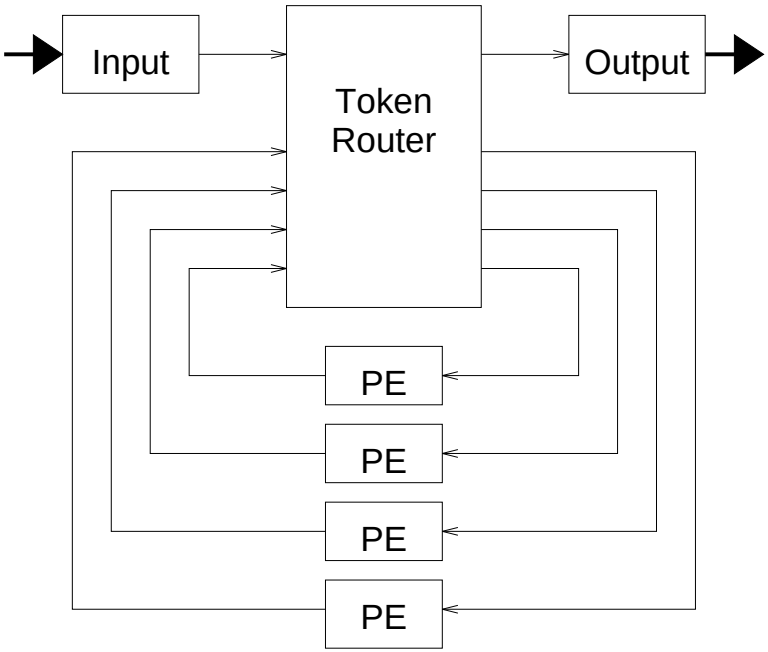
- Each PE is programmed to act as a node from our graph.
- The link switch is programmed with the graph topology, feeding the data tokens to their correct nodes.
- Where a data token must be delivered to several nodes, it is the link switch which copies it.



Programmable Dataflow Computer

We can reduce the requirement for seldom used communication links using a new token protocol.

- Each token carries with it a destination address.
- The token router ensures that tokens reach the correct PEs.



Dataflow Computers

In most real machines we cannot provide one PE per node in our graph.[2]

Mapping of nodes onto PEs:

- A de-activated node doesn't need a PE.

  A de-activated node is one which is waiting for one or more data tokens before firing.

- When a node is activated we must allocate a PE for it.

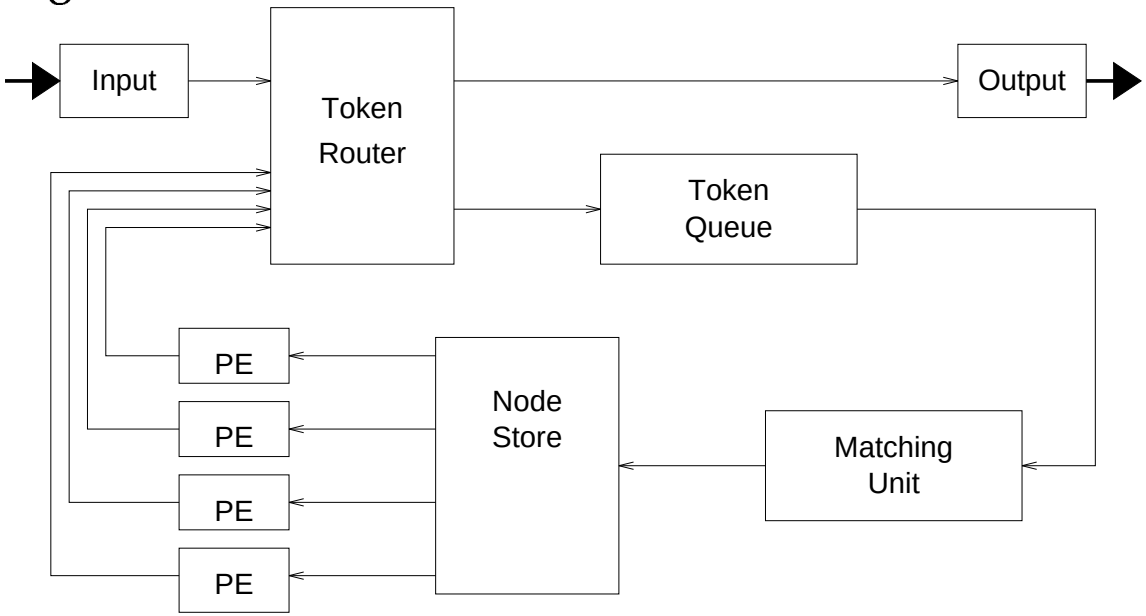
  This is done from the set of idle PEs.

- If there are no idle PEs, the activated node must be placed in a queue.

[2]In fact this would be inefficient in any application where there is not a constant stream of input data to keep the nodes busy.

Dataflow Computers

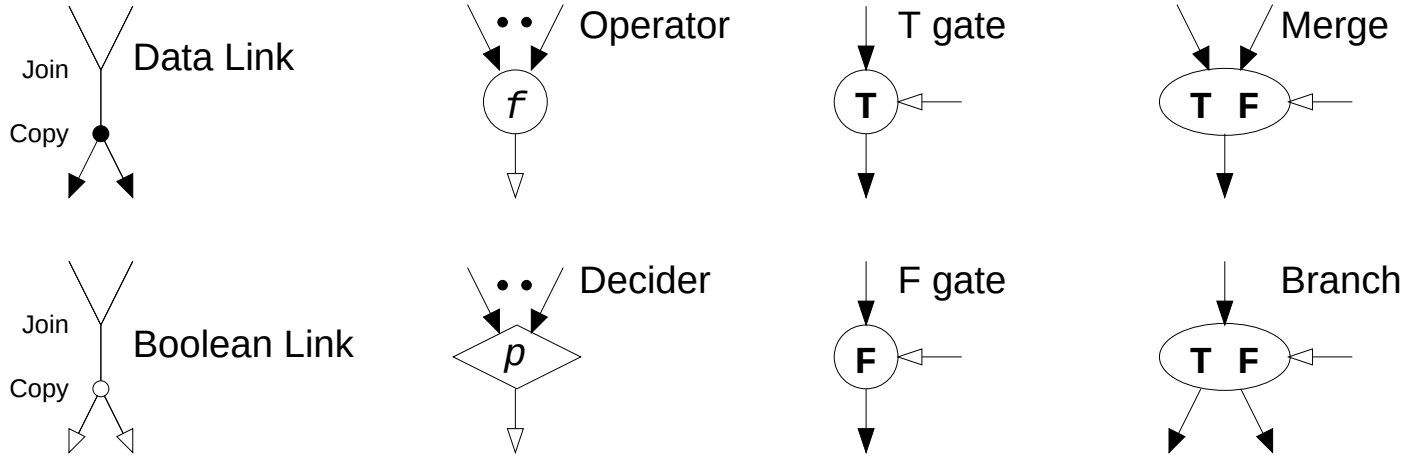
The following architecture is based on the Manchester Dataflow Computer:



- De-activated nodes are matched with data tokens in the matching unit.
- Activated nodes are stored in the node store awaiting a free PE.
- The token router deals with I/O and orders data tokens for placement in the token queue.

Dataflow Language Issues

To allow for decision making and iterative calculation, we enrich our dataflow graphs with further operators.



- **Data Link**[3]

  Carries general data values such as integer, real and complex numbers.

- **Boolean Link**

  Carries Boolean values for control purposes.

[3]Note the *join*, which passes all input tokens in some order, and the *copy*, which implicitly copies a data token to a number of destinations.

Dataflow Language Issues

- **Operator**

  Applies an operator, *f*, to its input values to produce its result.

- **Decider**

  Applies a predicate, *p*, to its data input values to produce a boolean result.

- **T gate(/F gate)**

  - Passes data if it receives a *true*(/*false*) value on its boolean input.
  - Discards data if it receives a *false*(/*true*) value on its boolean input.

- **Branch**

  Passes its data input to one of its two outputs dependent upon its boolean input.

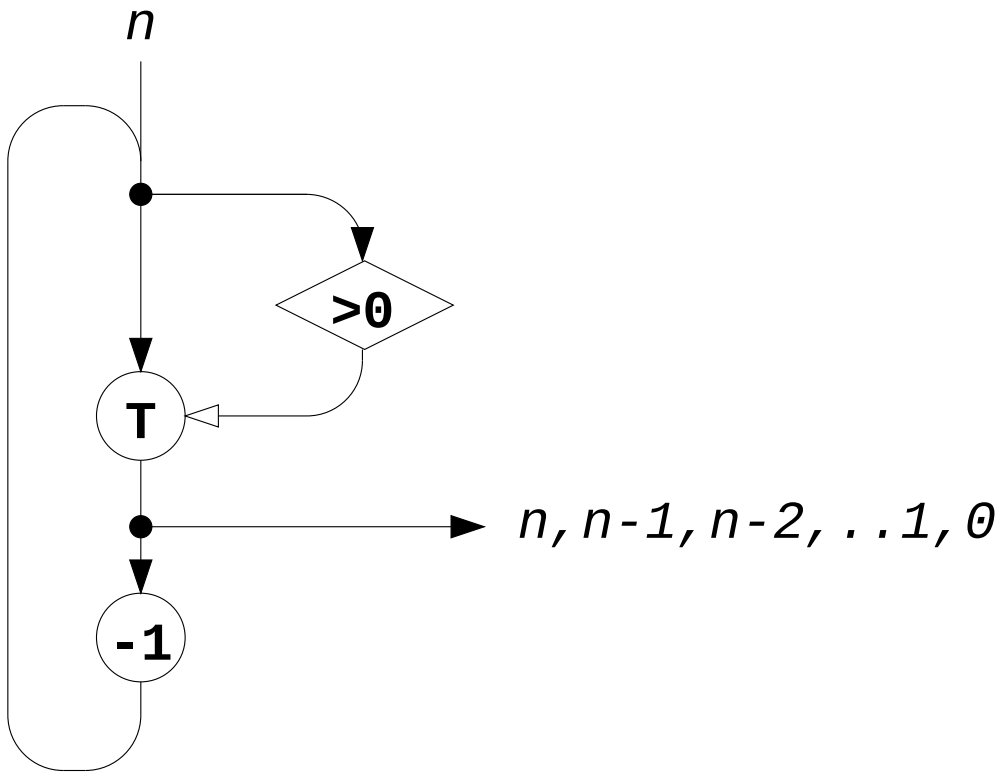
- **Merge**

  Passes one of its two data inputs to its output dependent upon its boolean input.

Other sets of operators exist, this set merely attempts to be self consistent.

Dataflow Language Issues

The following graph illustrates some of these operators, used to perform an iterative calculation.



Dataflow Language Issues

Dataflow graphs are usually generated from suitable languages. These are single assignment languages and declarative (zero assignment) languages.

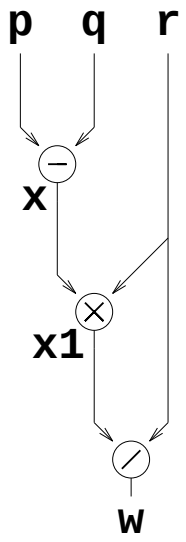
Taking single assignment languages (since these are simpler to understand), they obey the *single assignment* rule making them easy to convert into dataflow graphs.

Single Assignment Rule

Each variable may only occur once on the left hand side of an assignment statement.

```
x := p-q;
x := x*r;
w := x/r;
```

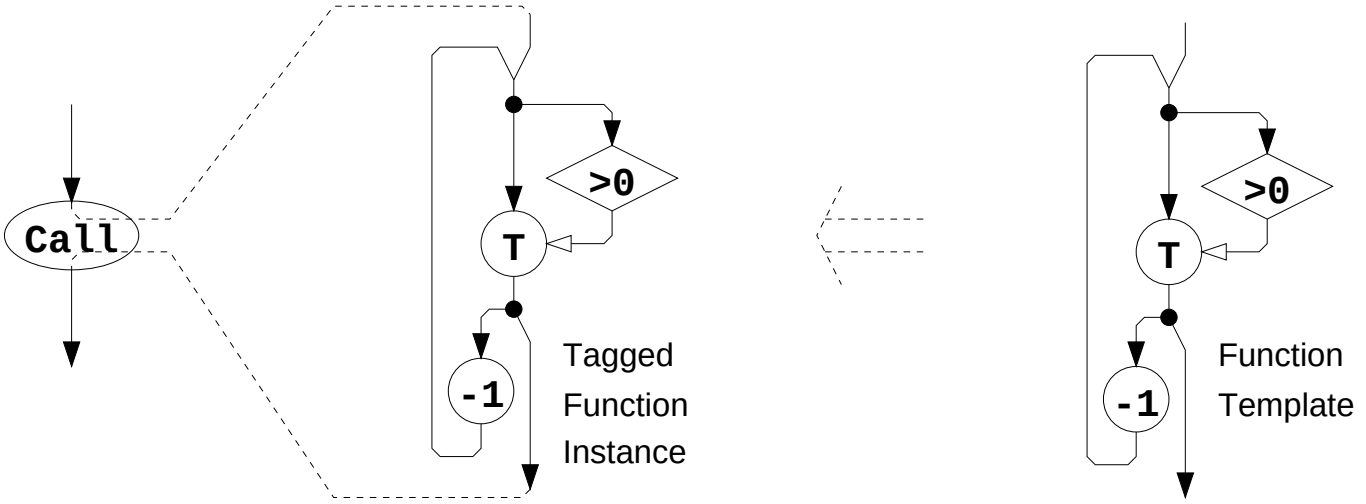
```
x := p-q;
x1:= x*r;
w := x1/r;
```



Other beneficial features include freedom from side effects and locality of effect.

Function Calls

- Expansion of function calls *in line*.
  - Inefficient, multiple copies of graph.
  - Precludes recursion.
- Dynamic instance creation.
  - Graph section is dynamically copied and destroyed on completion.
  - Nodes and tokens are tagged to distinguish between instances.



Dataflow Computer Issues

Dataflow computers provide an interesting alternative to parallel control flow computers.

While we avoid many problems associated with parallel control flow computers, we introduce new problems. These include:

- Copying of large data sets.
  - Each destination node requires a different token.
- Communications Bottleneck.
  - Due to the small granularity of the operations, the overhead for token transport is significant.
- Strange Languages
  - Few people are familiar with the languages, some concepts are very different from procedural languages.
  - It is suggested (by declarative programmers) that declarative programming can increase programmer productivity!