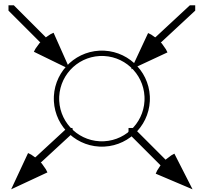
Topics in computer architecture

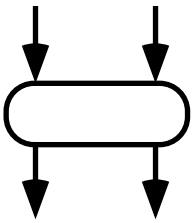
Data-driven nets

P.J. Drongowski
SandSoftwareSound.net

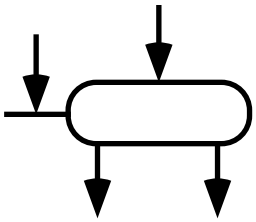
DDN cell types



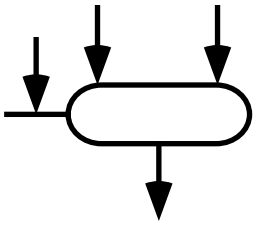
- Operator cell
- Conjunctive firing rule - token at all inputs
- Purely functional behavior - no side-effects
- Cell is labelled with function to be computed



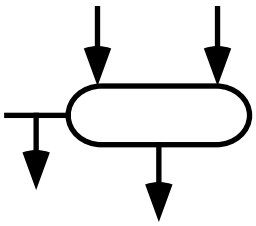
- Synch (synchronizer) cell
- Conjunctive firing rule - all inputs satisfied
- Passes each input to corresponding output
- One or more inputs (outputs)



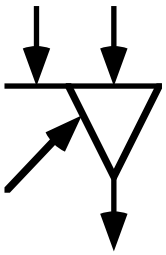
- Distribute cell
- Two inputs: control and input-value
- Conjunctive firing rule
- Copies input to selected output
- Numbered from 0 to N-1; left to right



- Select cell
- Firing set: control input and selected input
- Value of selected input is copied to output
- Inputs numbered from 0 to N-1; left to right



- Arbiter cell
- Firing rule: a value at any input
- Input value to is sent to output
- Index of selected input is output from control
- Inputs numbered from 0 to N-1; left to right



- Gate cell
- Initial state: Fire and copy value on gate input
- Condition true: Fire and copy feedback value
- Condition false
  - Value at gate input: Fire and copy gate input
  - Empty gate input: return to initial state

Arithmetic, relational, logical operators

- Arithmetic
  - `NEG` - negative
  - `ABS` - absolute value
  - `ADD` - addition
  - `SUB` - subtraction
  - `MUL` - multiply (extension)
  - `DIV` - divide (extension)
  - `MOD` - modulus (extension)
  - `MIN` - minimum of two values
  - `MAX` - maximum of two values
  - `10^N` - Shift left (multiply by power of 10)
- Relational
  - `LT` - less than
  - `GT` - greater than
  - `LE` - less than or equal to
  - `GE` - greater than or equal to
  - `EQ` - equal to
  - `NE` - not equal to
- Boolean (logical)
  - `NOT` - logical complement of Boolean value
  - `AND` - logical AND (extension)
  - `OR` - logical OR (extension)

- Notes
  - Boolean values are 0 (false) and 1 (true)
  - Operators can be applied to "vectors" if conformable
  - "Vector" examples
    - `NEG: ((-5)(-10)(20))` → `((5)(10)(-20))`
    - `ADD: ((1)(2)),((3)(4))` → `((4)(6))`
    - `MAX: ((1)(9)),((8)(2))` → `((8)(9))`
    - `NOT: ((1)(0)(0)(1))` → `((0)(1)(1)(0))`
    - `LT: ((1)(4)),((3)(2))` → `((1)(0))`
  - Similar to APL nested arrays

The Storage Model (TSM)

- Generalized tree structure discipline
- Alphabet
  - Digits = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
  - Punctuation = { ., –, •, (,) }
- TSM structure or *field*
  - All data between matching parentheses
  - Parentheses are special characters, not data
  - First character after (
    - is a data character, then the field is a *record*
    - is a), then the field is *empty* (null)
    - is a (, then the field is a *file* and has structure
  - Fields within a file can be records, files or a mixture
  - Fields within a file can be indexed
    - The first index selects a field within the file
    - If that field is a file, another index may select in it
  - Records may also be indexed (by character)
  - Example
    - (((A)((D)(E))(B))((F)(G))(C)(H)))
    - ((1)) ⇒ ((A)((D)(E))(B))
    - ((1)(2)) ⇒ ((D)(E))
    - ((1)(2)(1)) ⇒ (D)

Execution errors

- Kinds of execution errors
  - Input sent to nonexistent cell
  - Input sent to a nonexistent input of an existing cell
  - Illegal data item (e.g., nonconformable) may be received
- Generate an error token, ⊥ (not (), empty)
- All cells propagate error token
- Similar to Backus' "bottom preserving functions"

TSM operators

`UP` - Removes the outside set of parentheses of input
- Input must be a file with one field
- Examples
  - `((1234)) ⇒ (1234)`
  - `(((34)(12))) ⇒ ((34)(12))`

`DOWN` - Encloses the input in parentheses
- Examples
  - `(1234) ⇒ ((1234))`
  - `((3)(2)(2)(5)) ⇒ (((3)(2)(2)(5)))`

`SIZE` - Count fields or characters
- If input is a file, return number of fields in file
- If input is a record, return number of characters
- Examples
  - `(345) ⇒ (3)`
  - `((98)(47563)) ⇒ (2)`

`LEVEL` - Return true if input has no structure; else false
- Examples
  - `(9) ⇒ (1)`
  - `((((1)))) ⇒ (0)`

`IREAD` - Index TSM structure
- Left input is a TSM structure to be indexed
- Right input is a TSM access vector
- Indices must be greater than or equal to 1
- Example
  - Left: `((1)((2)((1)(1))))`
  - Right: `((2)(2))`
  - Result: `((1)(1))`

More TSM operators

`CAT` - Catenates two input structures

- Takes left and right TSM inputs
- Removes the parentheses from each
- It is illegal to catenate a record and a file
- Examples
  - `(12), (34) ⇒ (1234)`
  - `((1)(2)), ((3)(4)) ⇒ ((1)(2)(3)(4))`

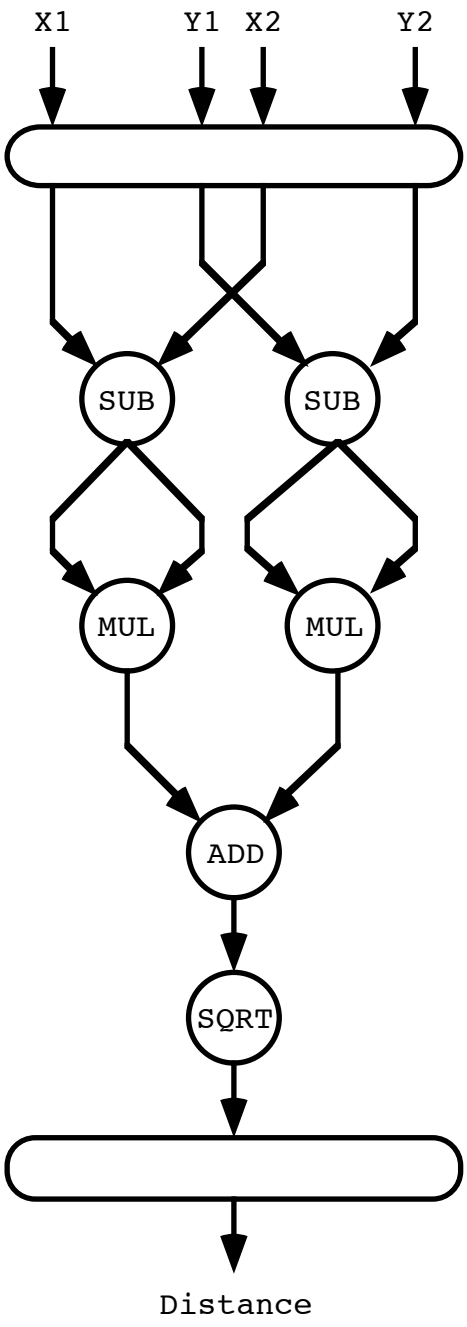
`DECAT` - Split a TSM structure into two parts

- Left input is a TSM structure
- Right input is an integer (N)
- Right result is the last N fields
- Left result is the first N fields
- Examples
  - `(123456789), (3) ⇒ (123456), (789)`
  - `((1)(2)(3)), (2) ⇒ ((1)(2)), ((3))`

`IWRT` - Indexed write

- Left input is a TSM structure to be modified
- Middle input is an access vector
- Right input is TSM structure to replace accessed field
- Examples
  - `((1)(2)(3)), ((2)), (7) ⇒ ((1)(7)(3))`

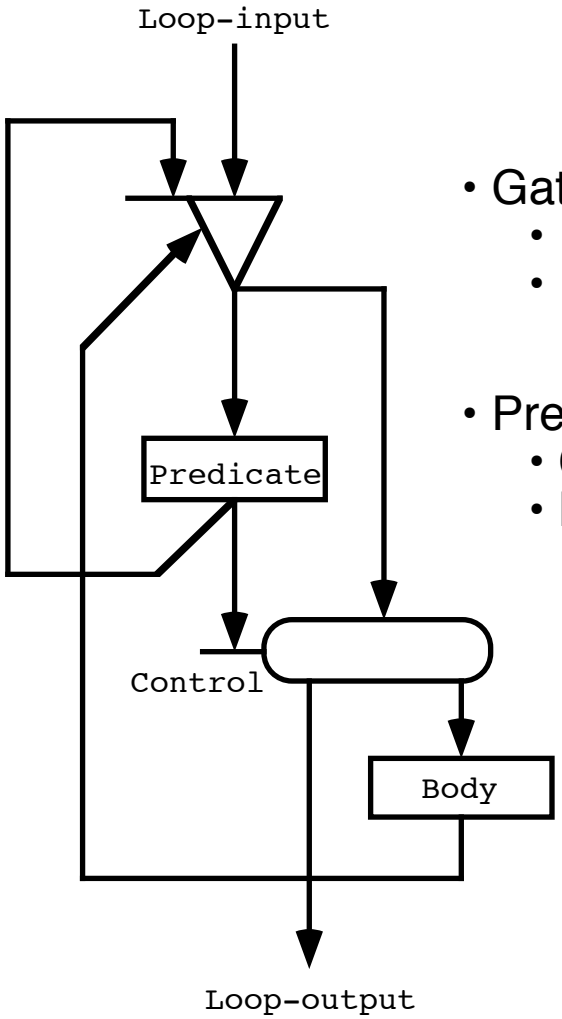
Process (subnet)



- Synchronized inputs
- (X1 - X2), (Y1 - Y2)
- Take square of differences
- Form sum of the squares
- Square root of sum of squares
- Synchronized outputs

- All arguments must be present before execution
- Results are not returned until al results are available

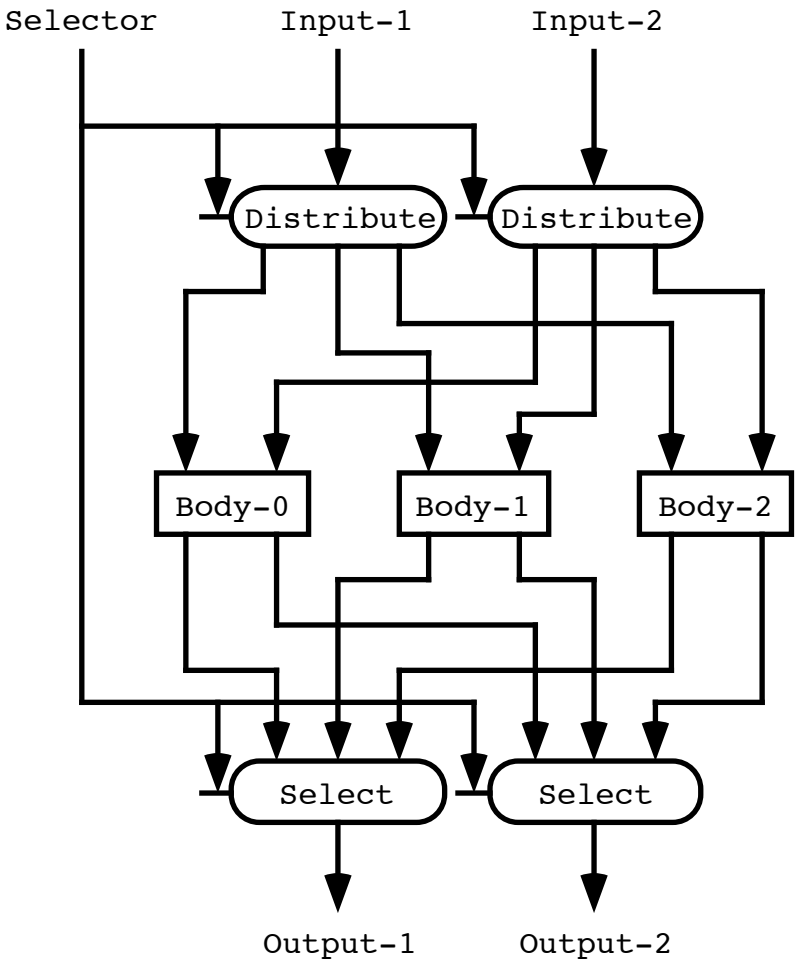
Loop form



- Gate cell chooses between:
  - Initial loop input value
  - Intermediate value
- Predicate forms condition for:
  - Gate control
  - Distribute cell
- Distribute
  - Produces loop output
  - Initiates body computation
- Body produces next loop value

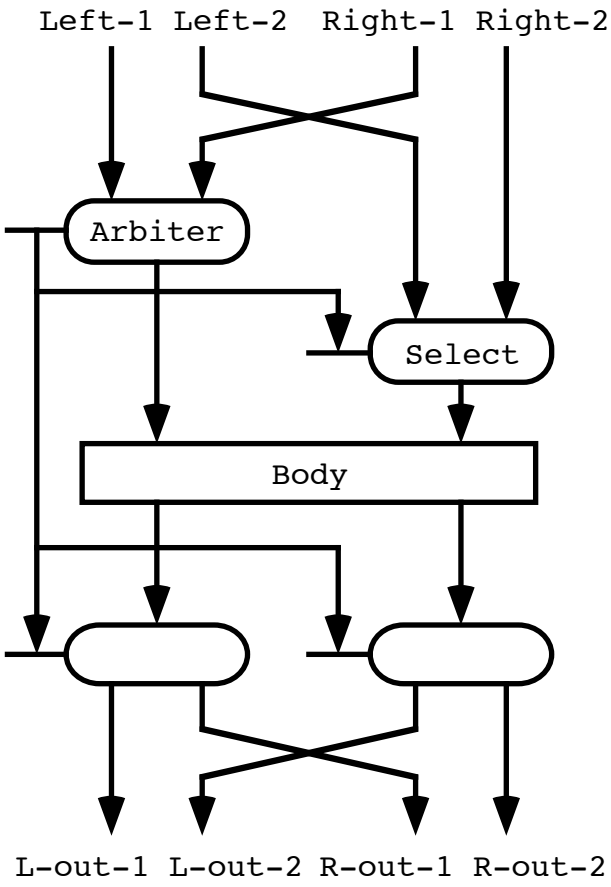
- One gate cell is required per loop input
- A body is required for each intermediate value
- Finally, a distribute cell is needed for each loop output
- The predicate can be copied to all gate and distribute cells
- Thus, a practical loop can and will look complicated!
- This loop terminates when the predicate is false (zero)

Case form



- Distribute cells route data inputs to case body nets
- Select cells choose results from a particular case
- Example has:
  - One case selection value (copied four times!)
  - Two input values
  - Three case bodies
  - Two output values
- Case should have "out of bounds" check

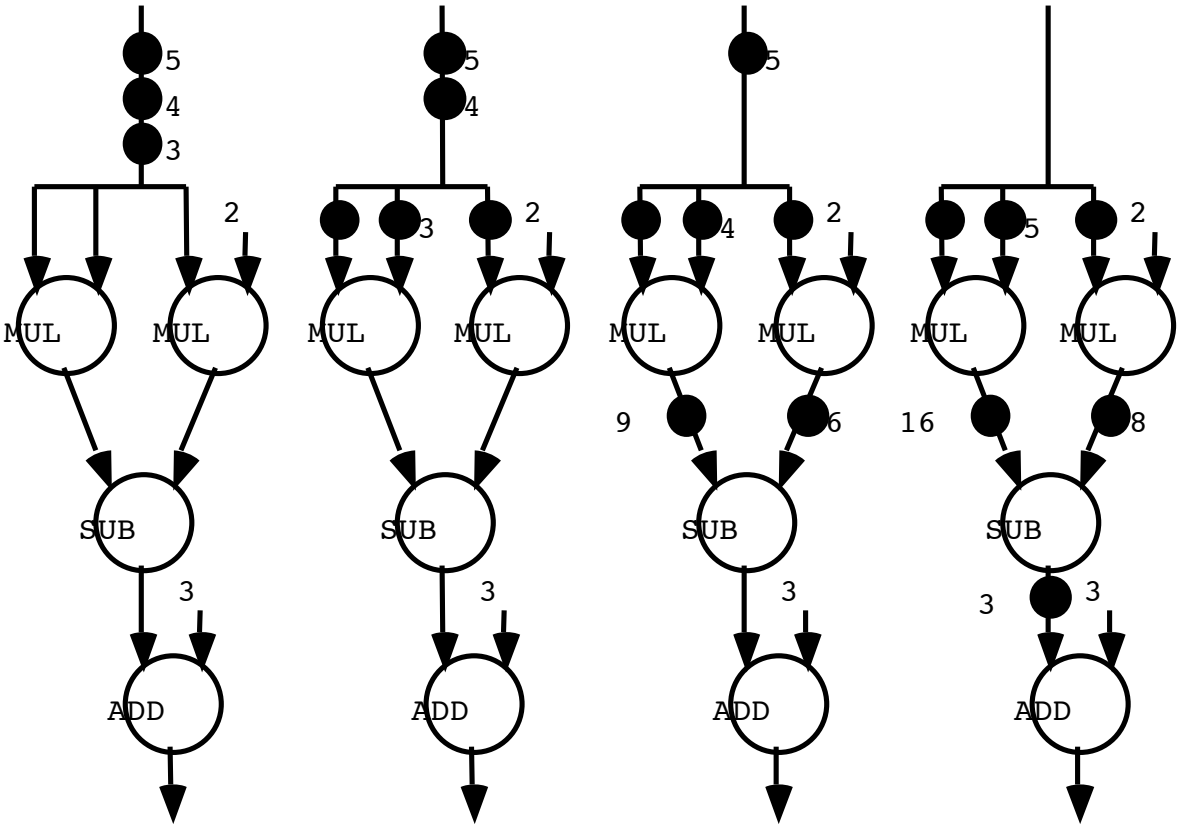
Share form



- Arbiter selects first caller
- Expand by adding select cells
- Results are routed to caller

- Used when a body is to be shared by several callers
- Body is executed sequentially
- Example has two inputs and two outputs per caller
- More callers will require:
  - Additional arbiter and select inputs
  - More outputs on the distribute cells at bottom
- This form assumes that inputs arrive in synchronized sets

Example: $X^2 - 2 \times X + 3$



- Constant values are regenerated as needed
- Execution is *determinate*
  - Arcs maintain first in - first out order
  - Multiple "fan in" to an input is disallowed
  - Cells are purely functional
- No arcs between `MUL` cells
  - No data dependency
  - Example of *horizontal* or *spatial* concurrency
- Arcs provide FIFO storage
  - Queue of values
  - Example of *temporal* concurrency or