

# A Smart Memory Design

## Concurrent-Processing Memory (CP memory)

By ChengPu Wang, Independent Researcher, USA

**Abstract:** A novel memory with

- simple processing power at each memory element;
- internal connectivity between memory elements.
- eliminates most streaming activities for data processing purpose on the data bus;
- general-purposed;
- easy to use, pin compatible with conventional memory;
- practical for implementation.

**Application to Parallel Operations:**

- Universal: matching, thresholding, insertion, deletion.  $\sim 1$ .
- Local (size  $M$ ): filtering  $\sim M$ , template matching  $\sim M^2$ , modeling  $\sim$  *single cell exchange count*.
- Global (size  $N$ ): sort, sum.  $\sim \sqrt{N}$  or  $\log_3 N$ .

# Conventional bus-sharing CPU/Memory Architecture

Advantages of the conventional bus-sharing CPU/Memory Architecture:

- Common and mature.
- Fit our Human logic well.
- Good for serial operations.

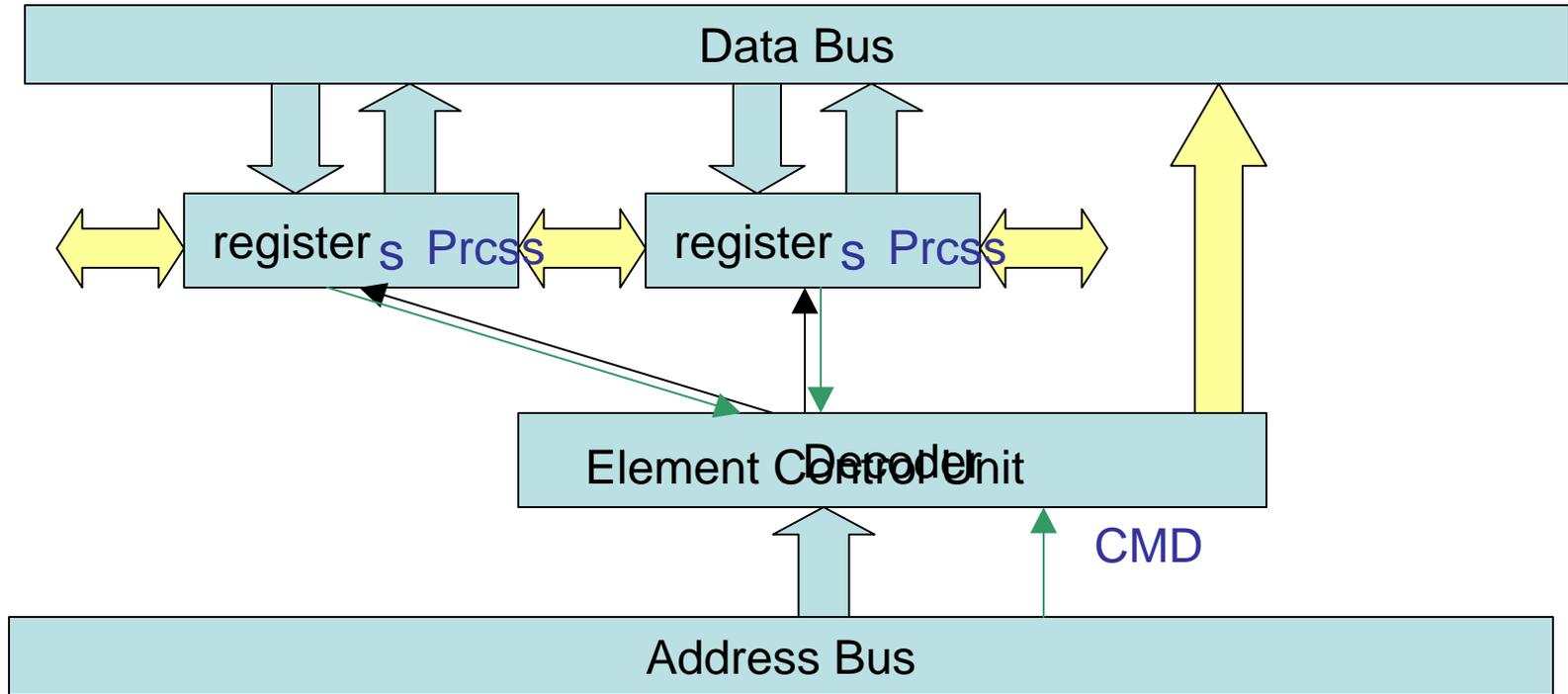
Not good for parallel operations, due to bus bottle-neck problems. For an example, 2D neighborhood averaging:

1. All pixels are streamed into CPU from the memory, multiple times.
  2. The neighbors are added inside the CPU serially.
  3. The results are streamed back to memory.
- Step 1 and 3 are necessary only due to serial implementation.
  - Step 2 may not be the most efficient algorithm.

Desire for parallel operations to be carried out:

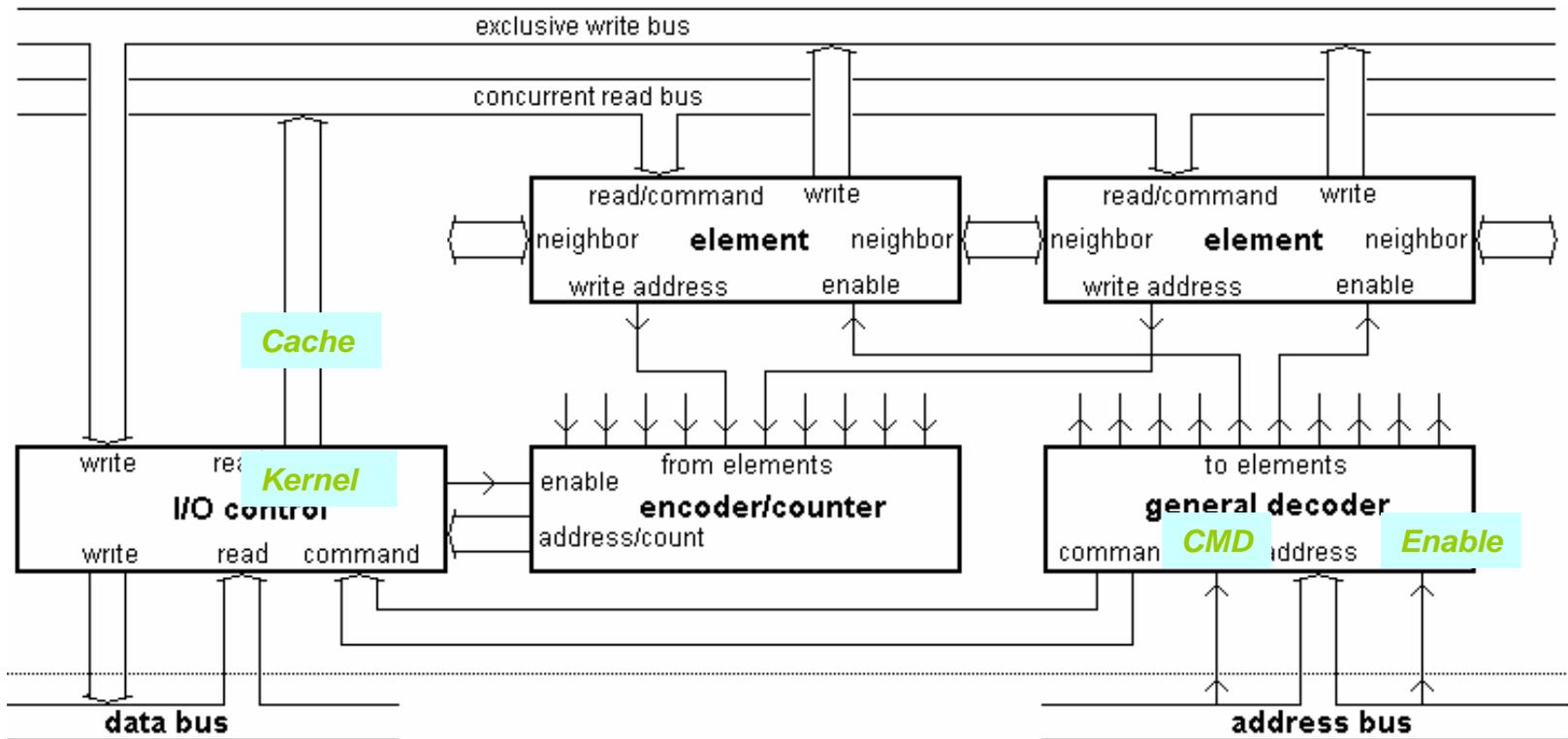
- Locally in a smart memory, which is pin-compatible with a conventional memory.
- Concurrently.

# Concurrent Processing Memory



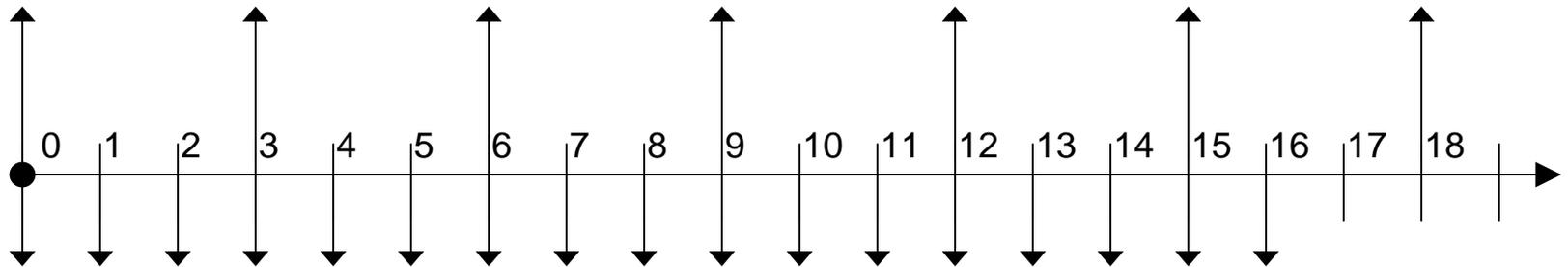
- Rule 1: Each element has one connection with the data bus, and has a unique address.
- Rule 2: Multiple elements can be required to write their addresses to the data bus concurrently. Each element asserts the line which connects it to the data bus.
- Rule 3: Each element is activated independently by the element control unit which is in use.
- Rule 4: Each element contains whether an instruction or address and data for the data bus.
- Rule 5: Neighboring elements are connected so that an element can read at least one element and write to at least one element.
- Rule 6: No connectivity between registers of each of its neighbors.
- Rule 7: The element control unit writes a predefined value to the data bus, indicating null.

# CP Memory System Architecture



- Element control unit: Encoder/Counter + General Decoder
- Enabled: by assigned high address
- Pin Compatible: LSB or MSB of the address bus as the CMD pin
- Internal cache: kernel (CP memory as co-processor)

## Address Logic in the General Decoder



- Carry-pattern Generator: inputs a carry number, activates corresponding (incremented) outputs, e.g. carry number is 3.
- Parallel Shifter: shift the carry pattern by an input address, e.g. 4.
- All-line Decoder: all lines below an end address are also activated, e.g. 16
- The two sets of outputs are AND-combined together to activate the corresponding elements.

Rule 9: All the outputs of a general decoder are activated if they correspond to the increment of the carry number starting at the input address and if they are equal to or less than the end address.

# Element Instruction Set

Addressing Registers: 0th (connected), 1st (operational), etc,

Minimal Instruction Set: a MUX and a comparator in each element

- *Conditionally execution* in respect to the values of its 0th and 1st registers.
- *Write address.*
- *Copy to its 1st register* from any one of: its own registers, the data bus, and its neighbor's 0th registers.
- *Copy its 1st register* to any of its own registers, or the data bus.
- *Reset* its 1st register to 0 or 1.

Enhanced Instruction Set: + an accumulator in each element

- *Negate* its 1st register.
- *Add* to or *subtract* from its 1st register any one of: its own registers, its neighbor's 0th registers, and the data bus.
- *Perform bit-shift operations* to its 1st register using the value of any one of: its own registers, its neighbor's 0th registers, and the data bus.

Internal Scaling: only code change is required;

- *Combine* a number of elements to hold one array item, using carry number;
- *Divide* one element by a number of array items, to save hardware.
- *Interleave* one element by a number of arrays, to save hardware.

# Use of Minimal Instruction Set

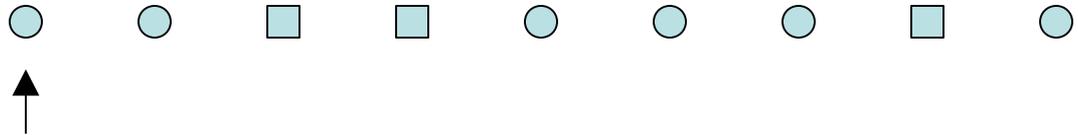
Insertion, Deletion: ~ 1 

- Buffer is always dynamic
- CP memory can be always closely packed



Matching:

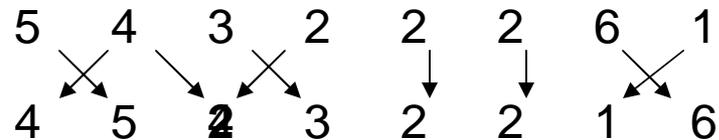
- to count match: ~1
- to find 1st match: ~1
- to enumerate all matches: ~ match count
- to construct histogram ~M, and to estimate/calculate sum.



Find Limit: two binary searches (for upper limit, and then for the maximum).

Sorting: e.g. into small to large

- direct comparison with neighbors
- to end sorting immediately;
- ~1 to sort all once;
- ~ N for simplest algorithm
- ~  $\sqrt{N}$  for an improved algorithm.

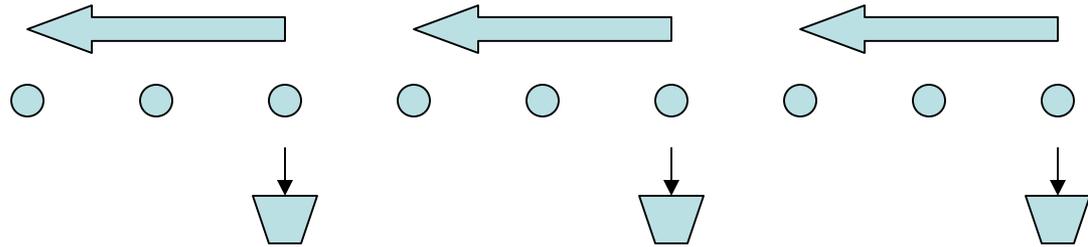


# Use of Enhanced Instruction Set

## Sum:

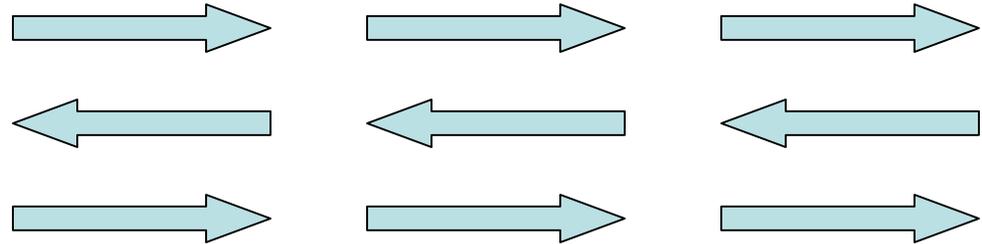
- parallel sum:  $\sim M$
- serial sum:  $\sim N / M$
- total:  $\sim \text{sqrt}[N]$

CPU-bus:



## Template matching: size M

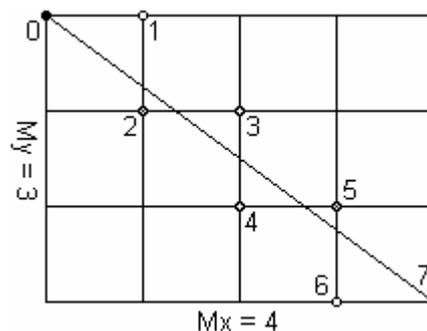
- $\sim M$  read, to all sections;
- $\sim M$  sum of diff, for all sections;
- $\sim 1$  shift to new position;
- $\sim M$  repeat for all positions.
- 1D  $\sim M^2$ , 2D  $\sim Mx^2 My^2$



Filtering: size  $M \sim M$ , e.g. Gaussian  $(1, 2, 4, 2, 1) = \text{original} + (1, 1, 1) \times 2$

## Line Detection: no floating math

- concurrent neighborhood counting;
- direct counting at  $0[\text{deg}]$  and  $90[\text{deg}]$ ;
- $\text{atan}[My / Mx]$ : messenger;
- angle resolution  $\rightarrow \{(Mx, My)\}$  set;
- concurrent messengers;



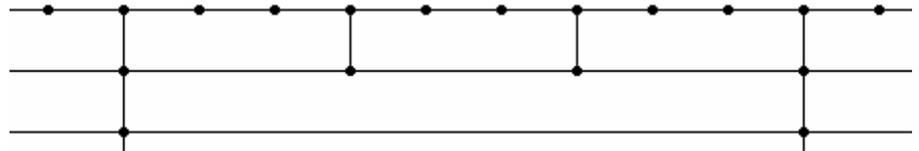
16	15	14	13	12
17	4	3	2	11
18	5	0	1	10
19	6	7	8	9
20	21	22	23	24

2-step messengers

# Connectivity

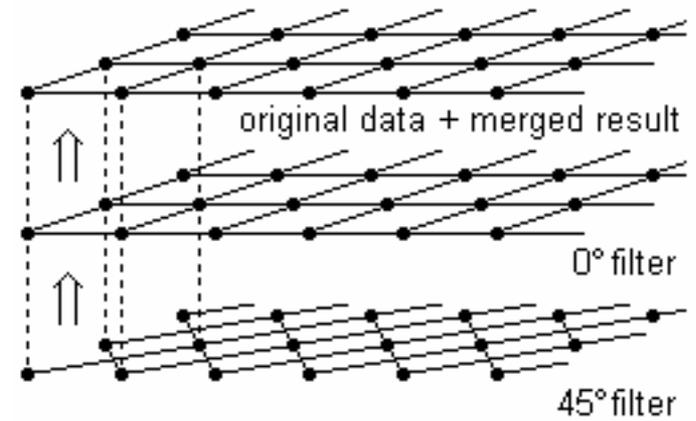
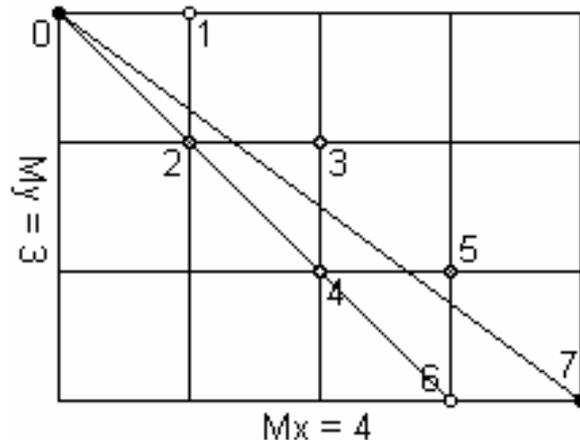
Use Long range connectivity, to reduce instruction cycle count for global operations further, e.g. sum:

- $\log_3[N]$  connection:  $\sim \log_3[N]$

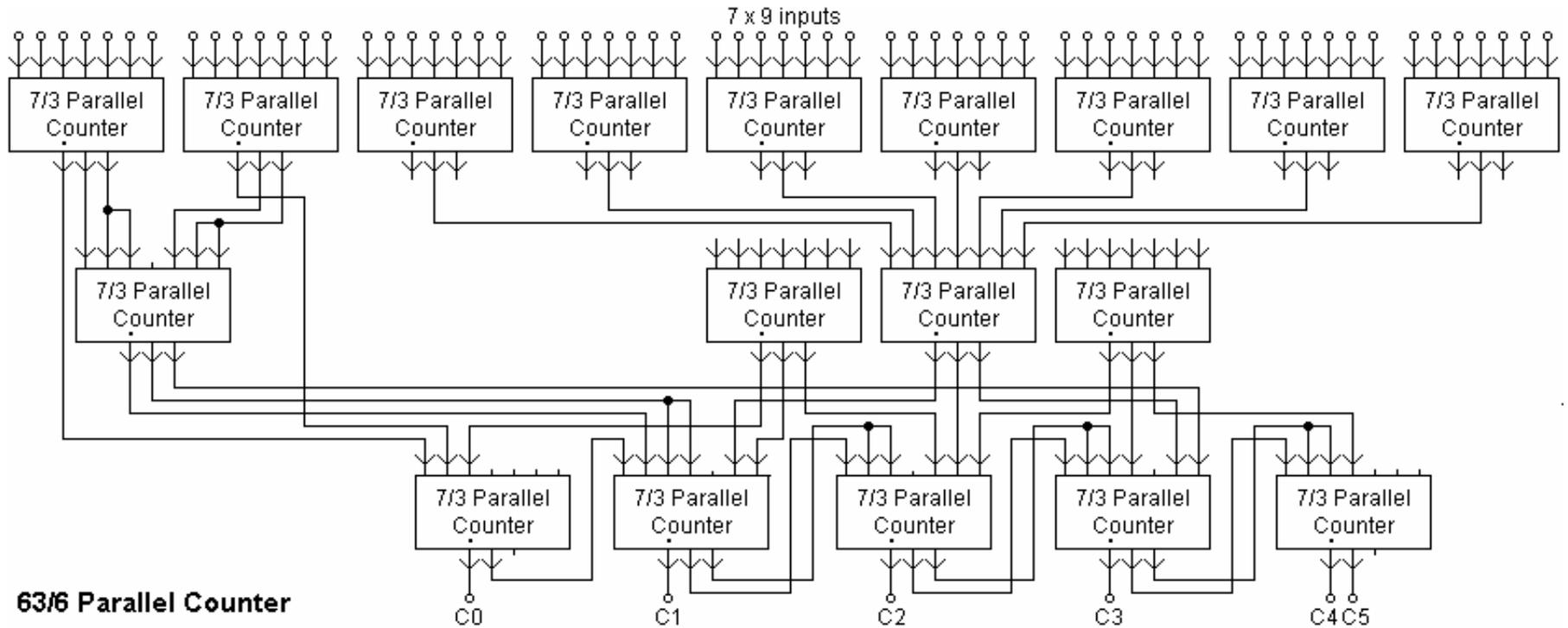


Connect M-dimension lattice into (M+1)-dimension lattice, e.g. line detection:

- Concurrent direct neighborhood counting in each plane, e.g. 0-7 and 0-2;
- SIMD pipeline;

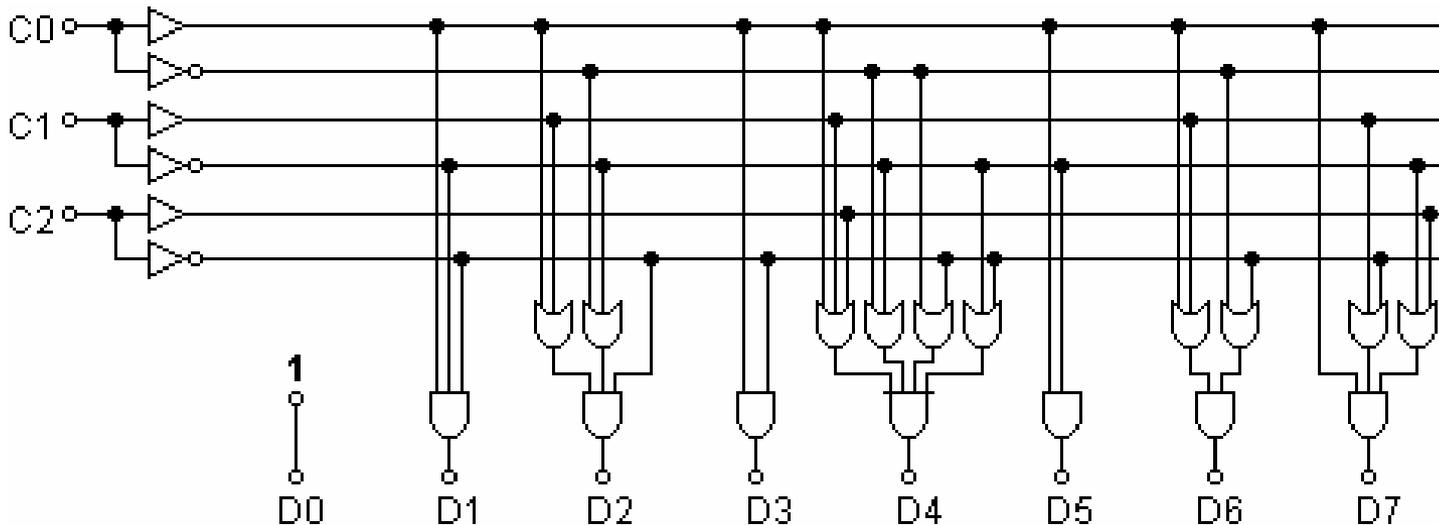


# Parallel Counter

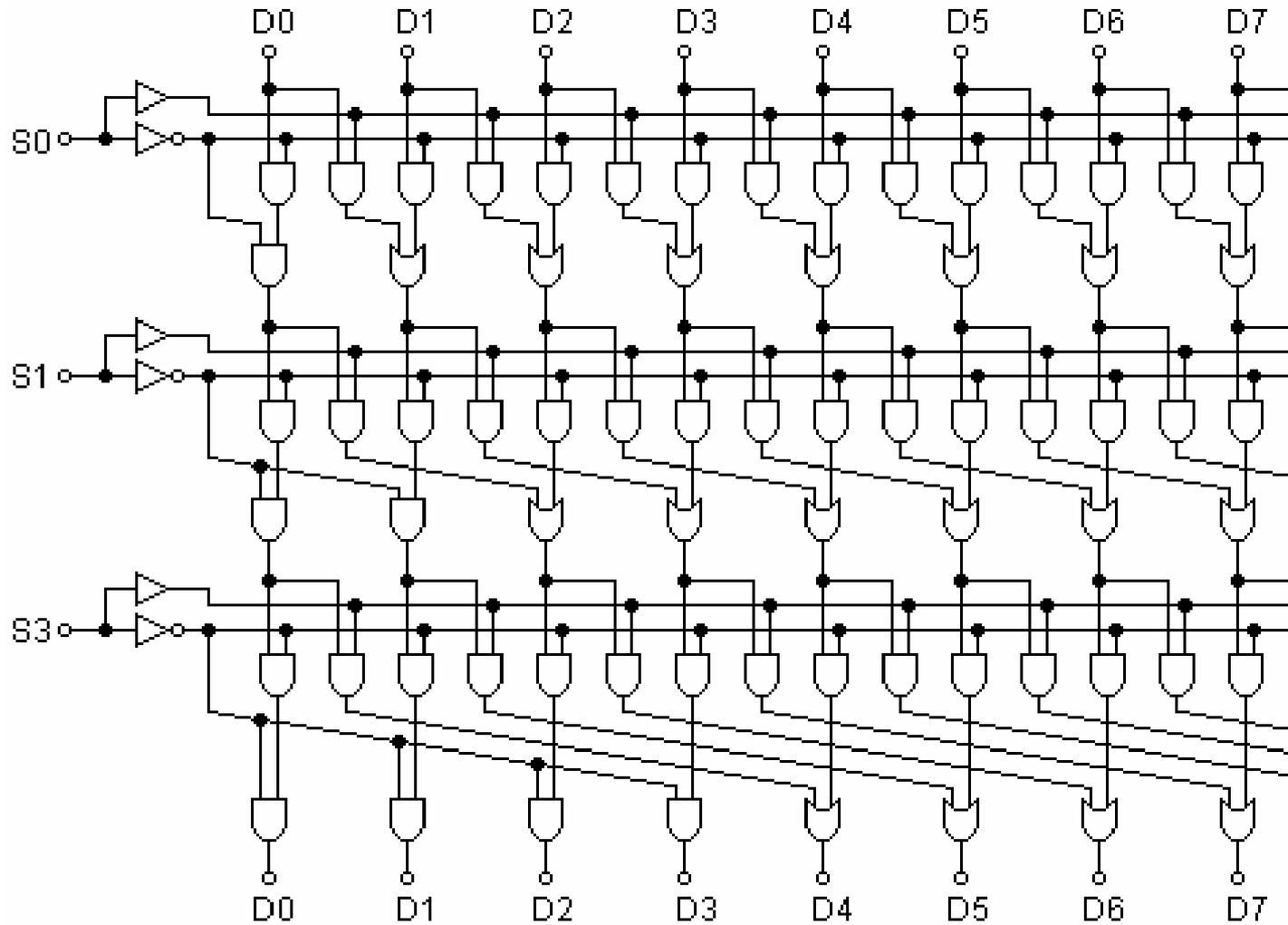


# Carry-pattern Generator

- $D0 = 1$
- $D1 = \neg C2 \neg C1 C0$
- $D2 = \neg C2 C1 \neg C0 + D1 = \neg C2 (C1 + C0) (\neg C1 + \neg C0)$
- $D3 = \neg C2 C1 C0 + D1 = \neg C2 C0$
- $D4 = C2 \neg C1 \neg C0 + D2 + D1 = (C2 + C1 + C0) (\neg C2 + \neg C1) (\neg C1 + \neg C0) (\neg C2 + \neg C0)$
- $D5 = C2 \neg C1 C0 + D1 = \neg C1 C0$
- $D6 = C2 C1 \neg C0 + D3 + D2 + D1 = (\neg C2 + \neg C0) (C1 + C0)$
- $D7 = C2 C1 C0 + D1 = (\neg C2 + C1) (C2 + \neg C1) C0$

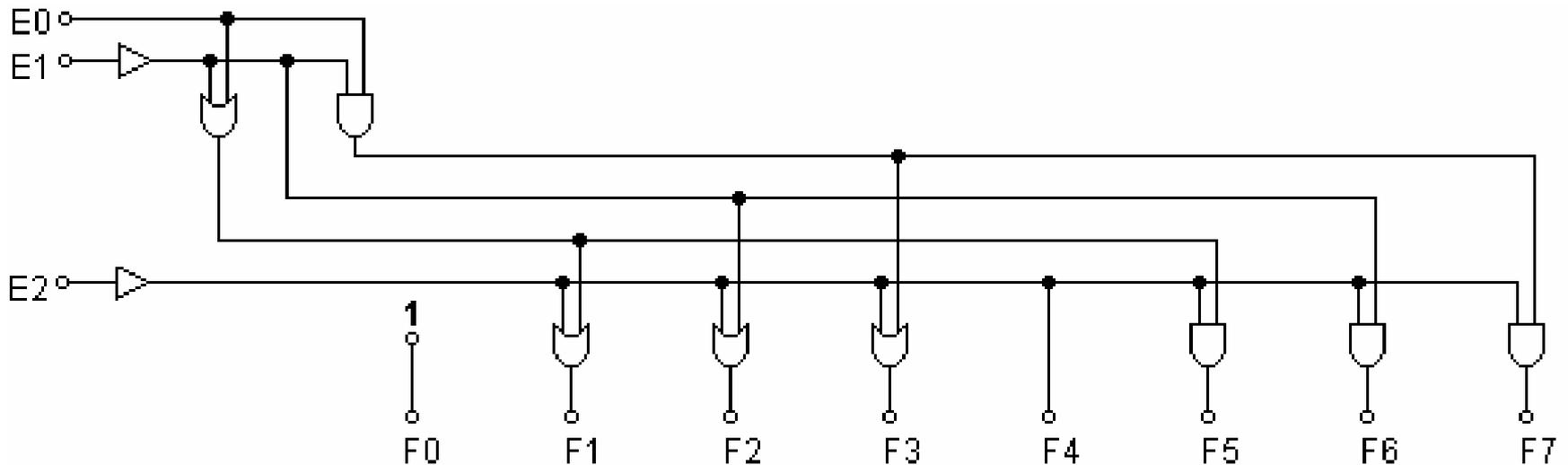


# Parallel Shifter



# All-line Decoder

- $F7 = E2 E1 E0 = E2 (E1 E0)$
- $F6 = F7 + E2 E1 !E0 = E2 E1 = E2 (E1)$
- $F5 = F6 + E2 !E1 E0 = E2 E1 + E2 E0 = E2 (E1 + E0)$
- $F4 = F5 + E2 !E1 !E0 = E2 = E2 (1)$
- $F3 = F4 + !E2 E1 E0 = E2 + E1 E0 = E2 + (E1 E0)$
- $F2 = F3 + !E2 E1 !E0 = E2 + E1 = E2 + (E1)$
- $F1 = F2 + !E2 !E1 E0 = E2 + E1 + E0 = E2 + (E1 + E0)$
- $F0 = F1 + !E2 !E1 !E0 = 1 = E2 + (1)$



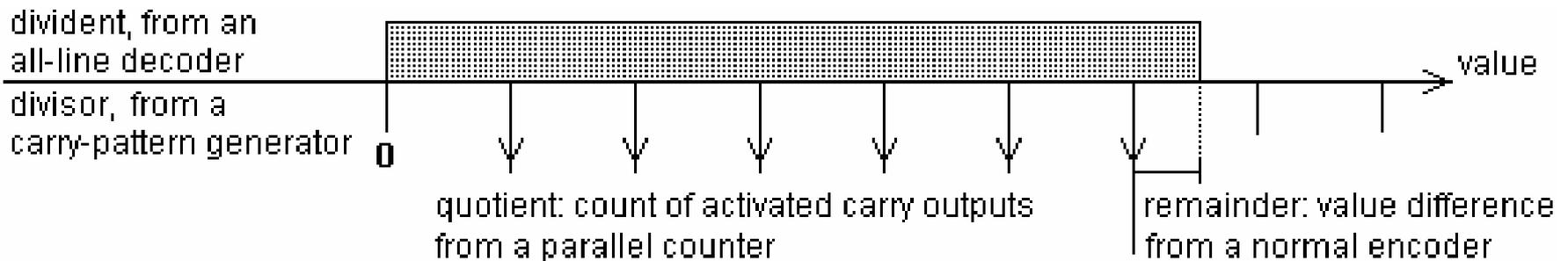
## Discussion:

### Other Applications:

- Video driver for animation.
- FFT, Matrix Math.

### Instant division:

Using Parallel Counter, All-line Decoder, and Carry-pattern Generator

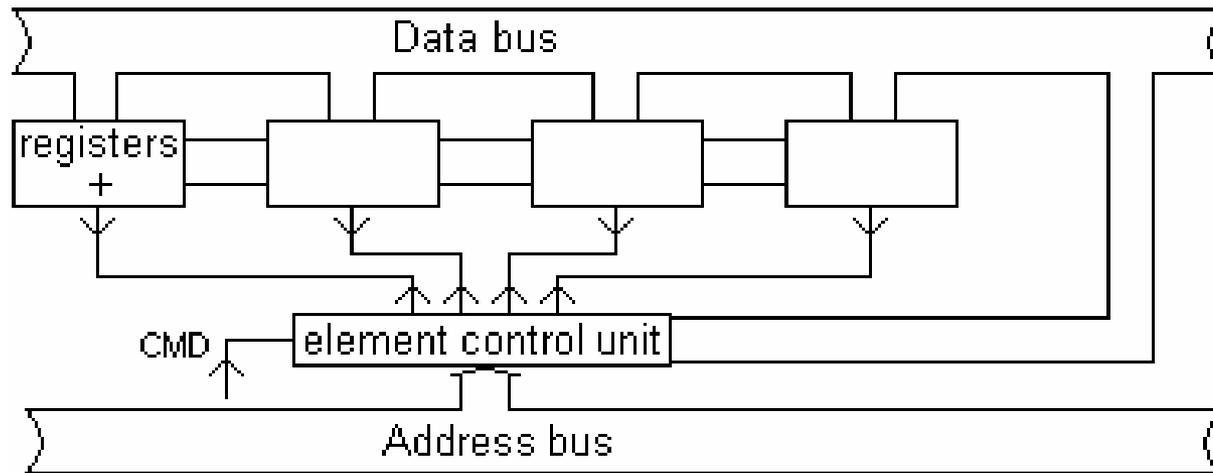


## Acknowledgement:

- Zhen Wang:
- Yingxia Wang:

Contact: [cpyx.wang@verizon.net](mailto:cpyx.wang@verizon.net)

# Summary



- Rule 1. Addressable.
- Rule 2. Exclusively read and write.
- Rule 3. Concurrent activation.
- Rule 4. Concurrent read.
- Rule 5. Concurrent address write.
- Rule 6. Multiple registers.
- Rule 7. Neighboring connectivity.
- Rule 8. Limited processing power: 5 instructions + 3 instructions.
- Rule 9.  $(\text{input address} + n * \text{carry number}) < \text{end address}$ .

# Improved Sort Algorithm ~ $\sqrt{N}$

1. Determine the best sorting order: the items to be sorted count should be less than  $N / 2$ .
2. Use the 1st algorithm until the items to be sorted is less than  $M$  in  $\sim N / M$  instruction cycles: (2, 5, 4, 2, 2, 3, 1, 6).
3. Find the 1st position to be sort from left whose left is not larger than right: (2, 5, 4, 2, 2, 3, 1, 6).
4. Move the item to the correct position: (**1**, 2, 5, 4, 2, 2, 3, 6)
5. Repeat Step 3 and 4.
6. Find the 1st position to be sort from right whose left is larger than right: (1, 2, 5, 4, **2**, 2, 3, 6).
7. Insert the left item to the correct position: (1, 2, 5, **2**, 2, 3, **4**, 6).
8. Repeat Step 6 and 7: (1, 2, 2, 2, 3, 4, **5**, 6).

Step 2 to 8 takes  $\sim M$  instruction cycles.

The total instruction cycles count is  $\sim \sqrt{N}$ .