Arithmetic Mean

Data Output

- Design a sequential circuit that computes the mean M of k n-bit numbers stored in registers
  - *e.g.*, accessing a RAM or register file with k addresses
  - To save on hardware, you can only use one n-bit adder and have a single read port RAM

No. of numbers

width of numbers

- Algorithm Pseudocode:

①
```
S = 0
for i = 0 to K-1
S = S + R[i]
end for
M = S/K
```

②
```
S = 0
for i = (K-1) to 0
S = S + R[i]
end for
M = S/K
```

③
```
M = 0
for i = K-1 to 0
M = M + R[i]/K
end for
```

Aside: Counter Variable

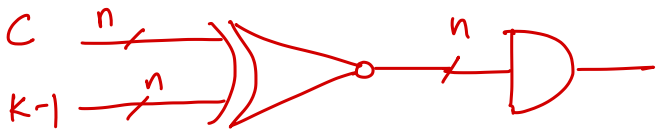
equality detectors ✓ XNOR.

- Many sequential hardware algorithms utilize counters
- If both work, is there a preference?
  - How to implement $C = k - 1$ check?

$C = C_{n-1}\ C_{n-2} \cdots C_1\ C_0$

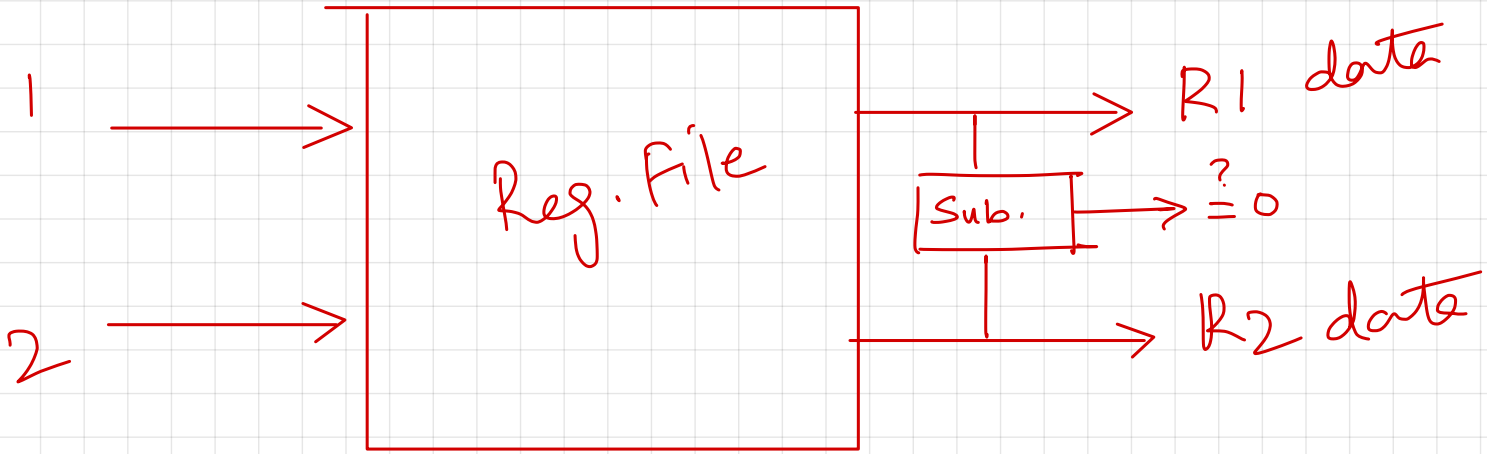
$K-1 = (K-1)_{n-1}\ (K-1)_{n-2} \cdots (K-1)_1\ (K-1)_0$

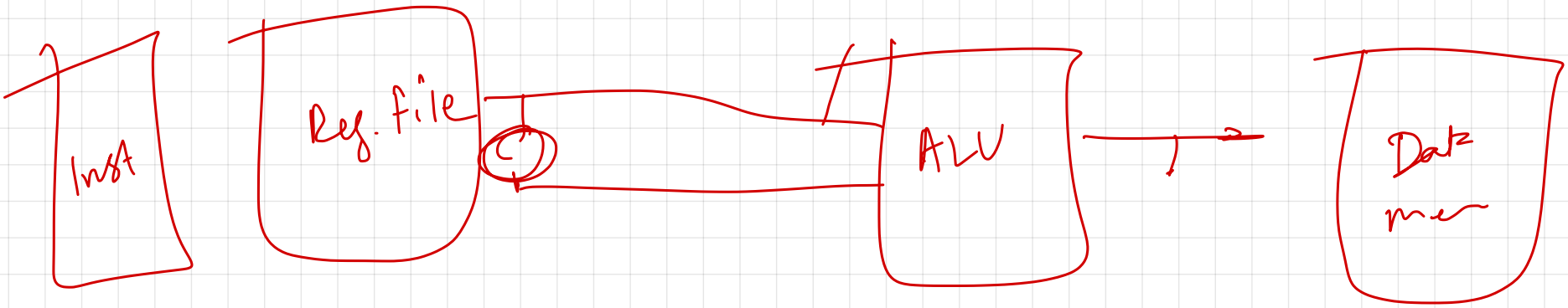
$[C - (K-1)] \stackrel{?}{=} 0$



  - How to implement $C = 0$ check?

Better


1
2
Reg. File
R1 data
Sub.
?
=0
R2 data


Inst
Reg. file
ALU
Data
mem

Arithmetic Mean Specification

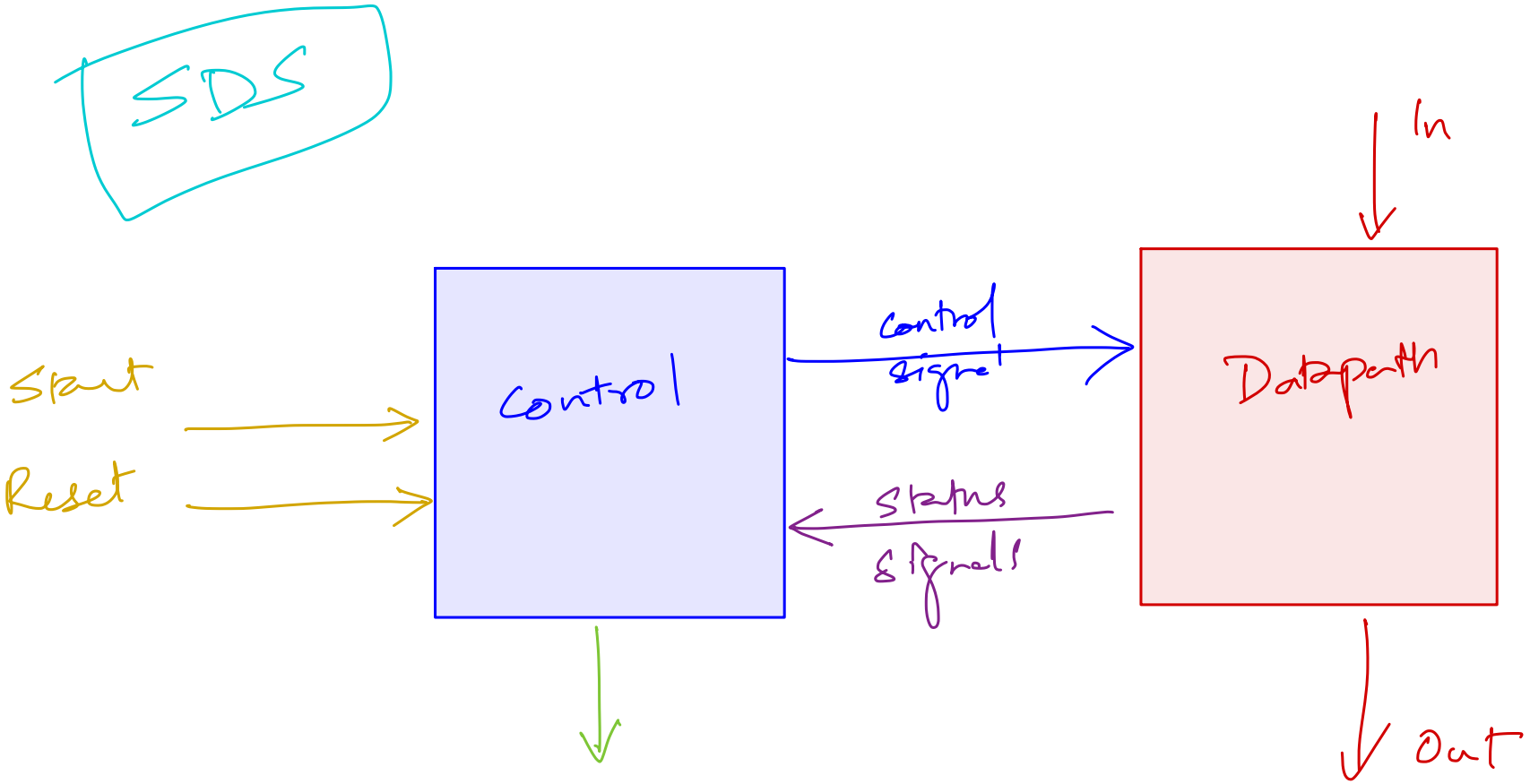
- Datapath
  - A k-address *register file* (only using `r_addr` and `r_data`)
  - Reg file address stored in $\lceil \log_2(k) \rceil$ *down-counter* A
  - Sum stored in register S
  - An n-bit *divider* circuit, as discussed last lecture

A Reg[A]
i Reg[i]

- Control
  - Inputs $Start$ and $Reset$, outputs $Ready$ and $Done$
  - Status signals: A_zero, Div_done, Div_ready
  - Control signals: Load_regs, Add, Divide, Decr_A

all bits of A ? = 0

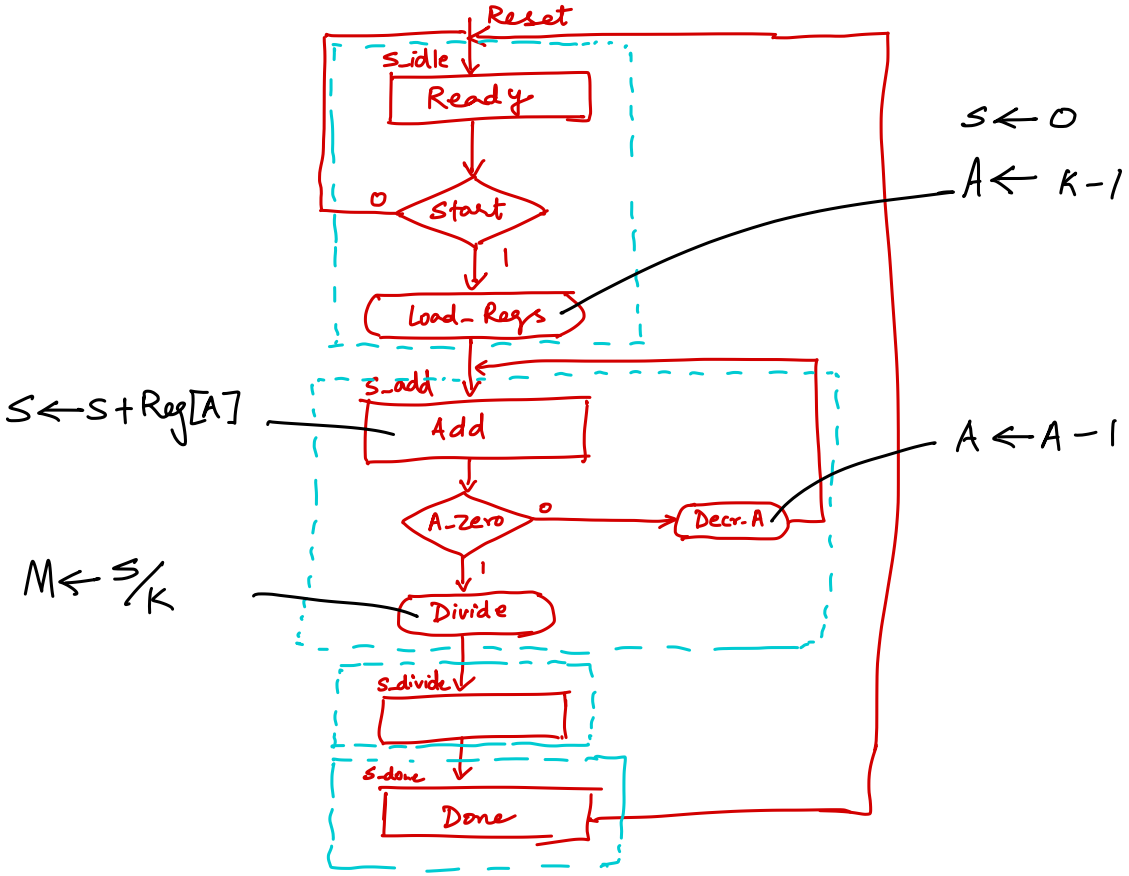
Status indicators.


SDS
In
Control
Datapath
Start
Reset
control
signals
status
signals
Out

Arithmetic Sum (ASMD Chart, Initial)

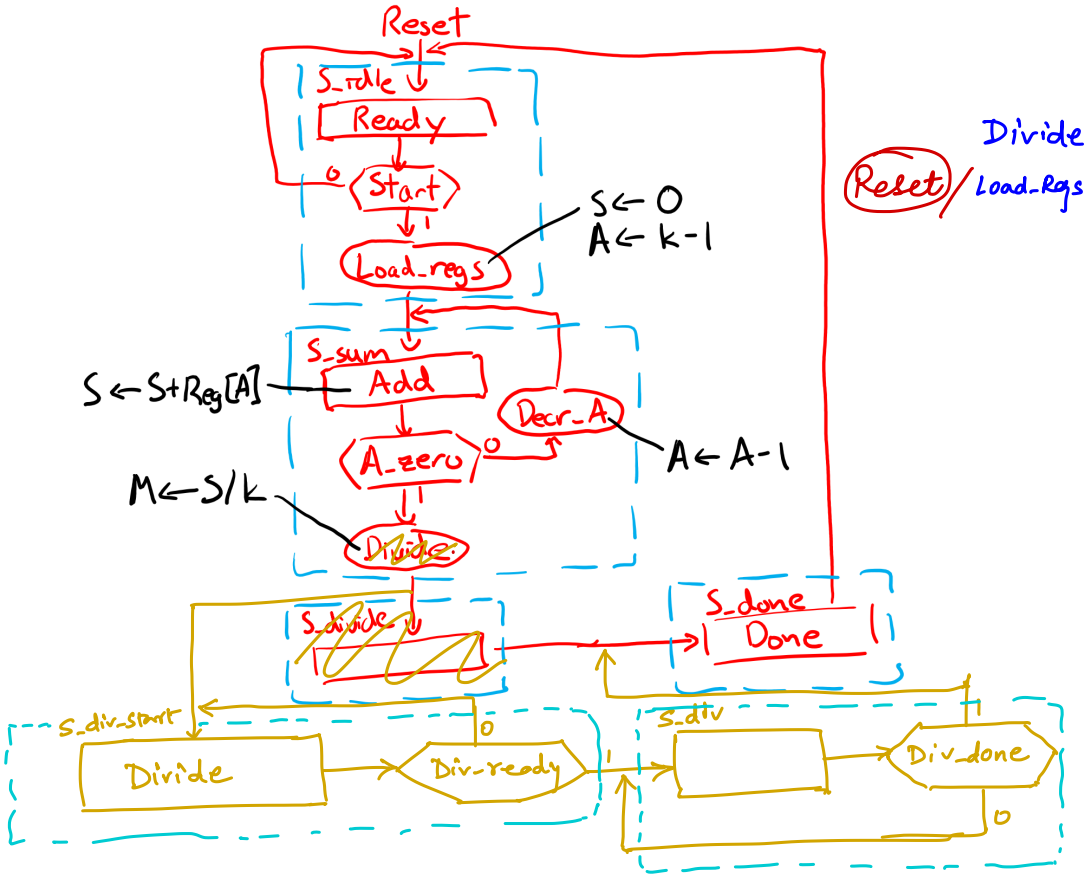
- For now, ignore the details of the divider circuit

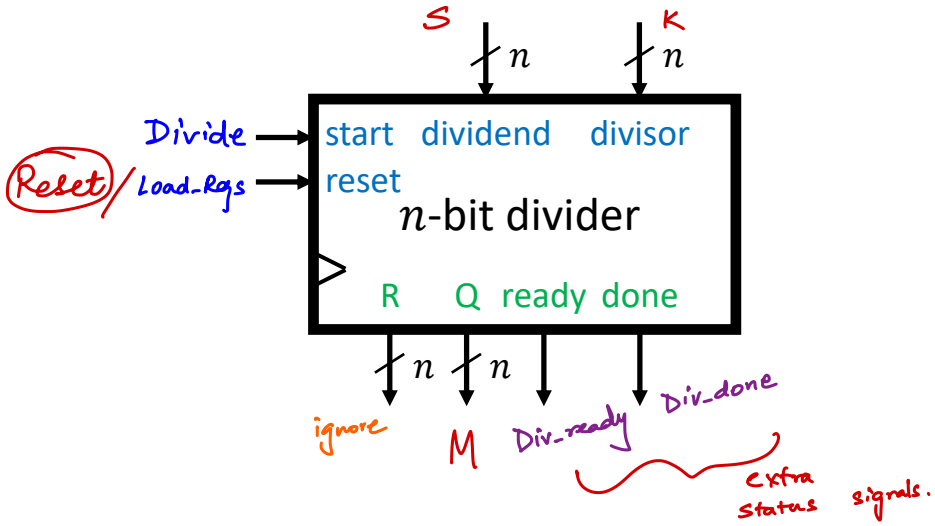
```
S = 0
for i = (K-1) to 0
S = S + R[i]
end for
M = S/K
```



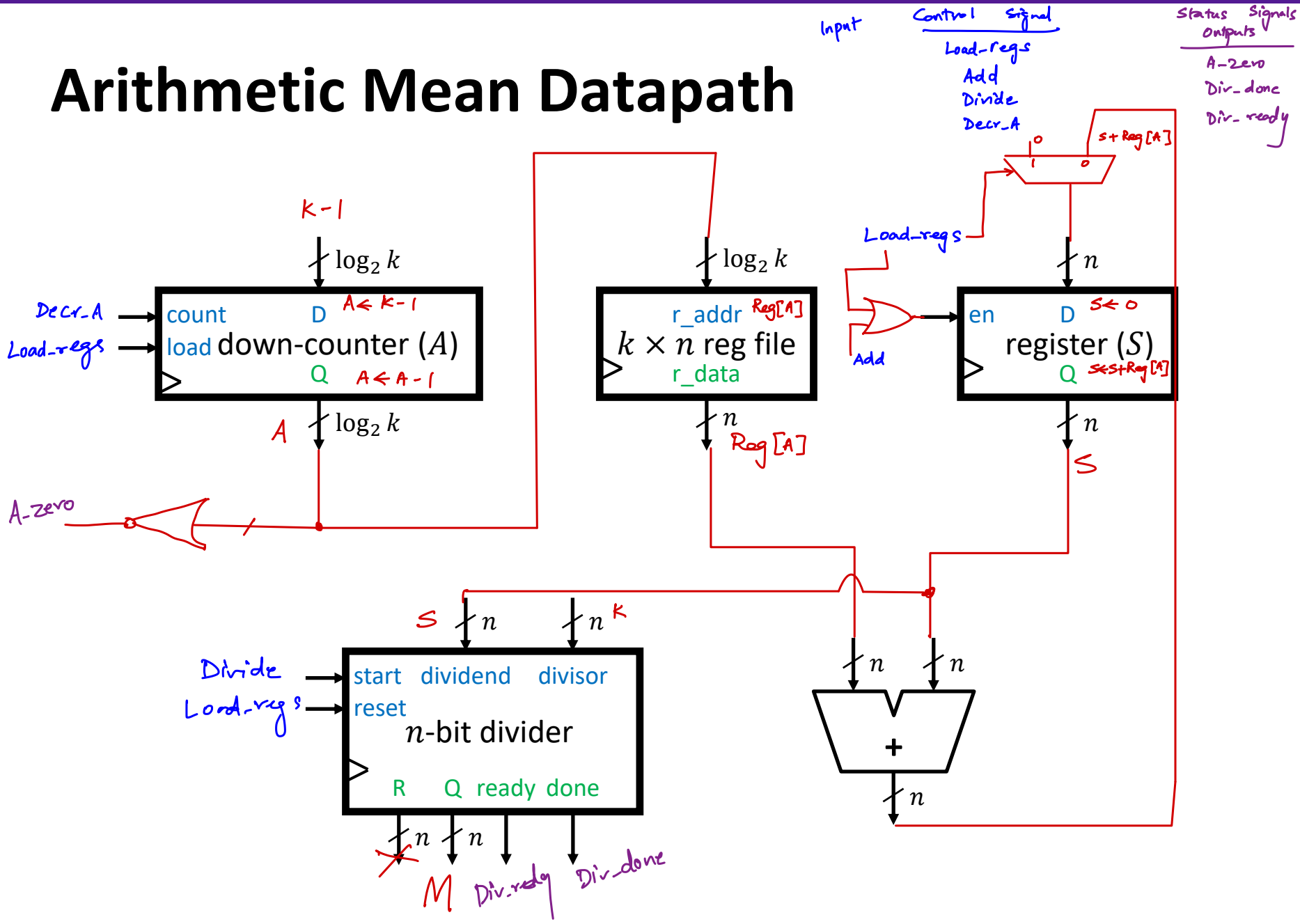
Arithmetic Sum (ASMD Chart)

- Fix your ASMD chart based on the divider circuit:





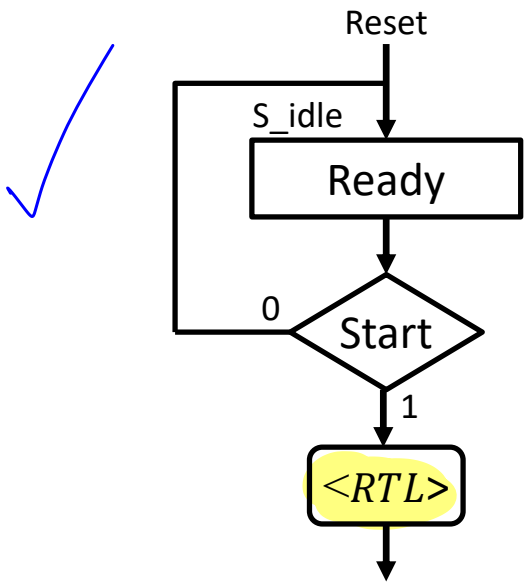
Arithmetic Mean Datapath

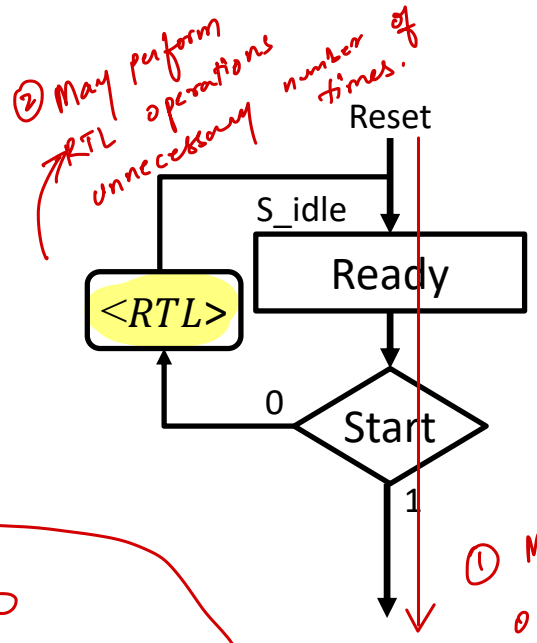


Technology Break

Aside: Load Loops

- For *some* initialization operations, you can get equivalent behavior from either the (1) outgoing edge or the (2) looping edge:

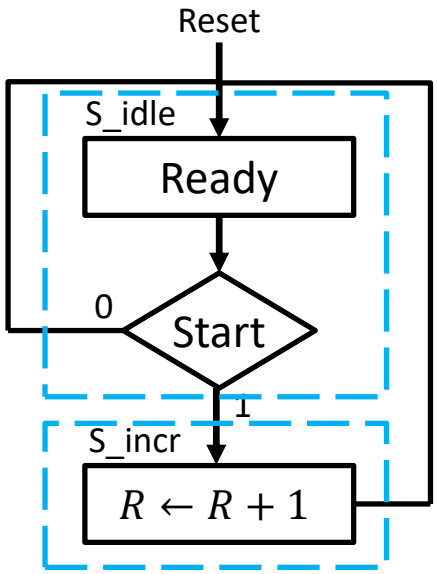




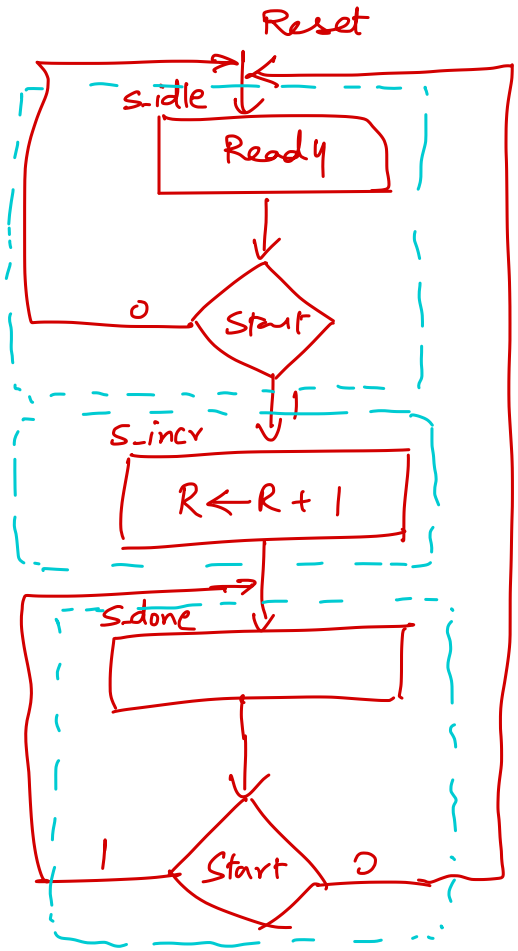
M ← 0
S ← 0
B ← divisor
A ← k-1

Aside: Start Loops

- What happens if we forget to de-assert *Start*?



- Fix: Start de-assertion loop.



Sorting Algorithm

Single port Read and Write

- Design a circuit to sort k n-bit numbers stored in a set of registers in ascending order

Algorithm:

```
for i = 0 to k-2 do
    A = Reg[i]
    for j = i+1 to k-1 do
        B = Reg[j]
        if B < A then
            Reg[i] = B
            Reg[j] = A
            A = Reg[i]
        endif
    endfor
endfor
```

init_i, i_done, incr_i, init_j, j_done, incr_j, Load_A, Load_B, B_lt_A, store_B, store_A, Swap.

Example ($k = 4$): $i \to 2$, $j \to 3$

i	j	A	B	R[0]	R[1]	R[2]	R[3]
0	1	3	7	3	7	1	0
0	2	3	1	3	7	1	0
0	3	1	0	1	7	3	0
1	2	7	3	0	7	3	1
1	3	3	1	0	3	7	1
2	3	7	3	0	1	7	3
				0	1	3	7

Sorting Algorithm Specification

- Datapath
  - A k-address *register file* (assume only 1 port)
  - Two $\lceil \log_2(k) \rceil$ *up-counters* i and j
  - Two registers A and B
  - An n-bit *comparator* circuit to check for $B < A$

- Control
  - Inputs $Start$ and $Reset$, outputs $Ready$ and $Done$
  - Status signals: i_done, j_done, B_lt_A
  - Control signals: incr_i, incr_j, init_i, init_j

Load_A, Load_B, Store_A, Store_B.

A ← Reg[i] B ← Reg[j] Reg[j] ← A Reg[i] ← B.

Sorting Algorithm Specification

- Datapath
  - A k-address *register file* (assume only 1 port)
  - Two $\lceil \log_2(k) \rceil$ *up-counters* i and j
  - Two registers A and B
  - An n-bit *comparator* circuit to check for $B < A$

- Timing Notes:
  - RTL operations in a state occur on the *next* clock trigger
  - Can `i` ← `x` and `A` ← `Reg[i]` be done simultaneously?
  - Can `Reg[i]` ← `B` and `Reg[j]` ← `A` be done simultaneously?
  - Swap operations *must* be done sequentially