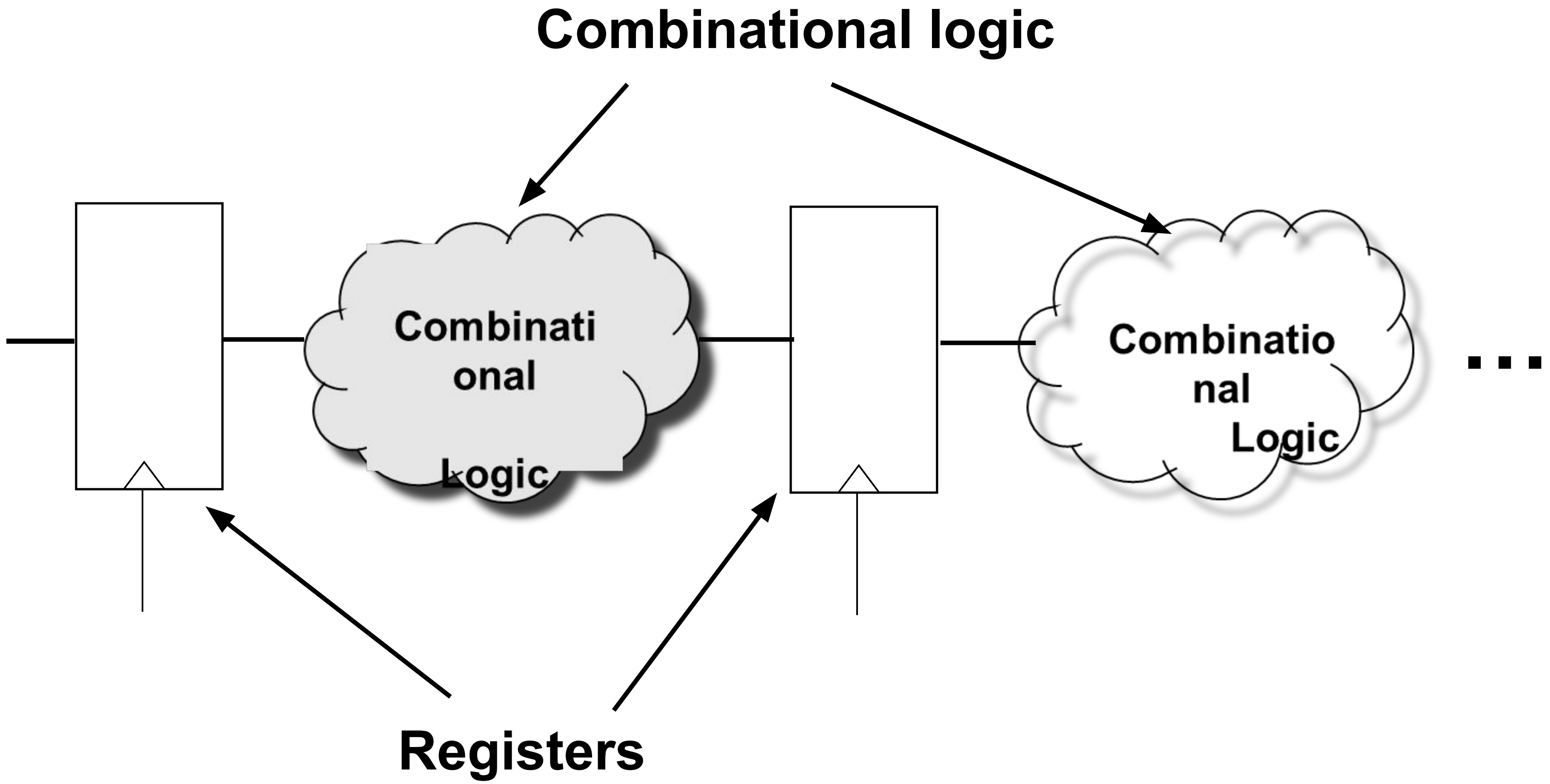
Sequential Logic Design

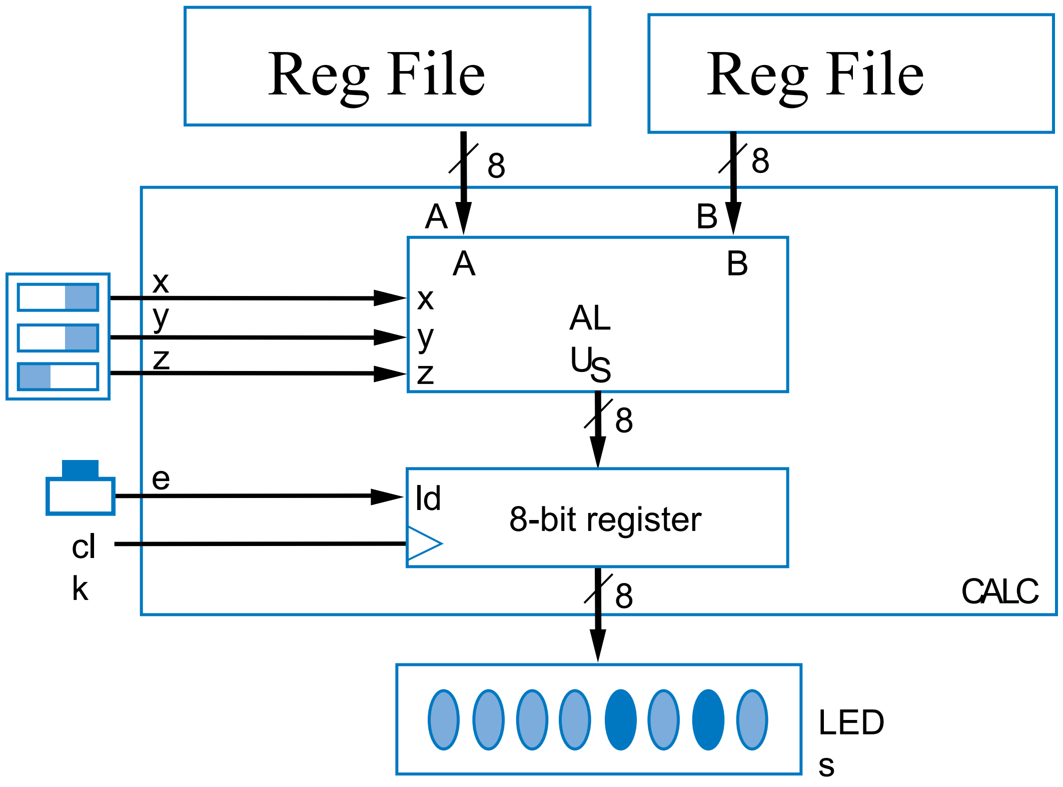
Register file, Multi-function Register, FSM, Controller

Register Transfer Level Design Description



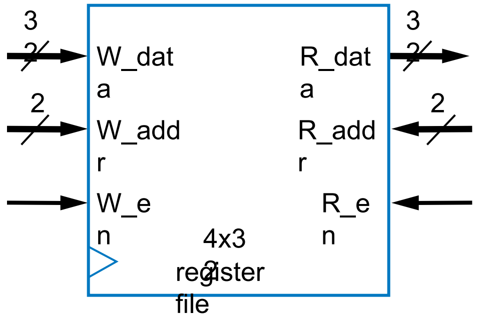
ALU Example:

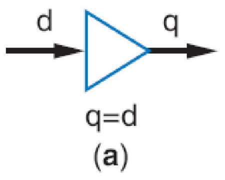
- Design using ALU is elegant and efficient

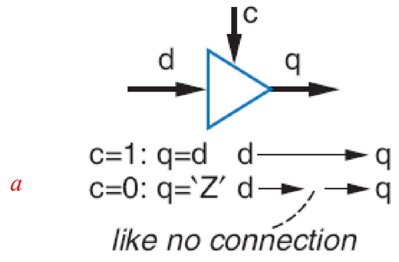


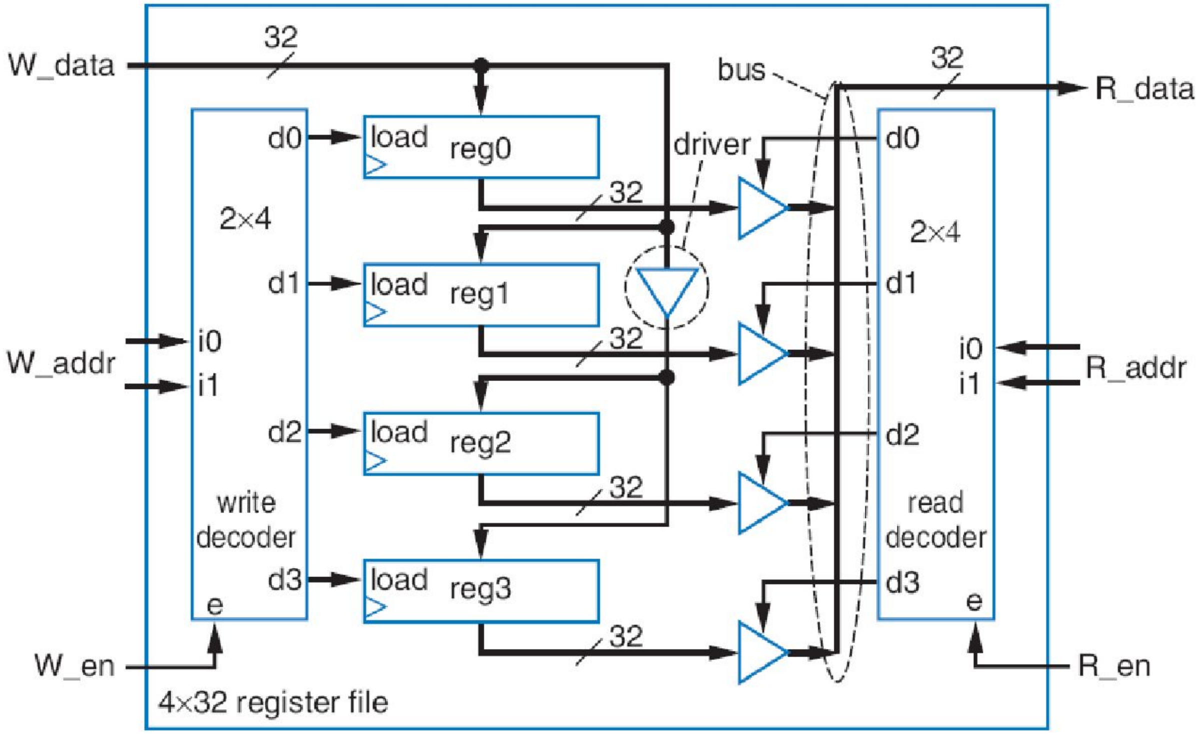
Register File

- Instead, want component that has one data input and one data output, and allows us to specify which internal register to write and which to read



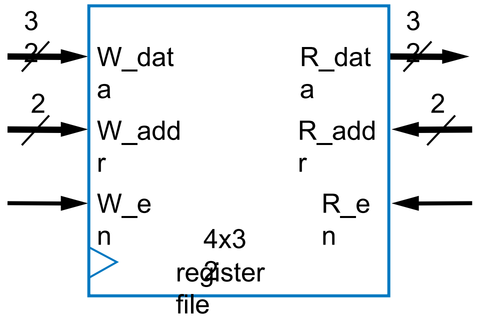




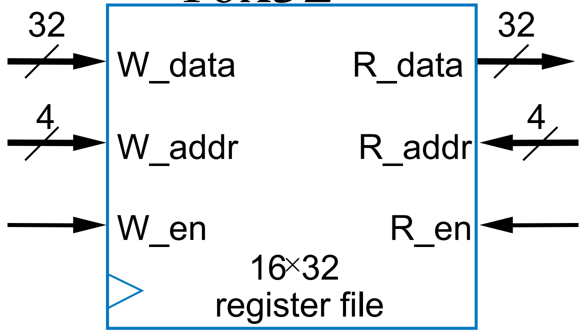


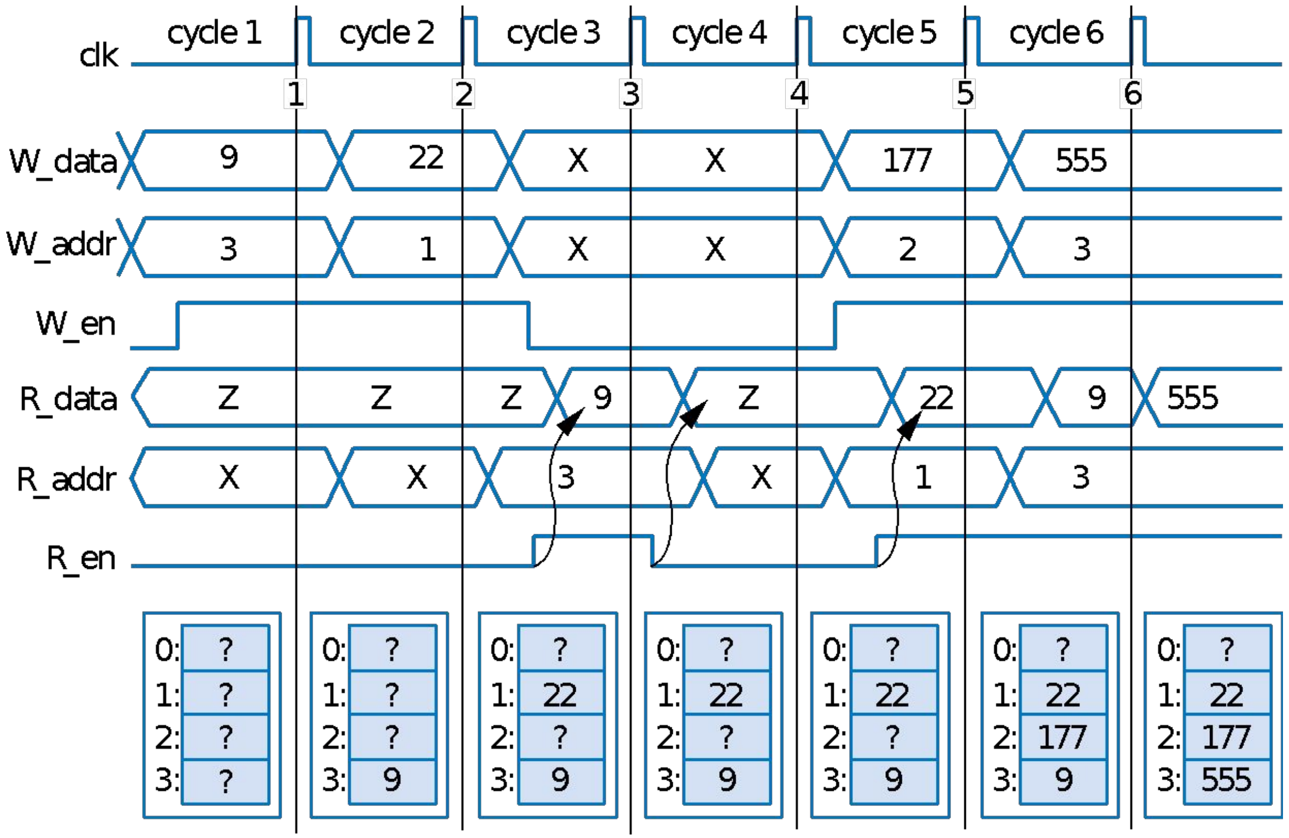
Register File Timing Diagram

- Can write one register and read one register each clock cycle
  - May be same register

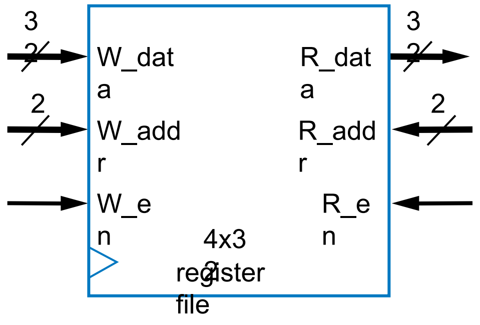


16x32

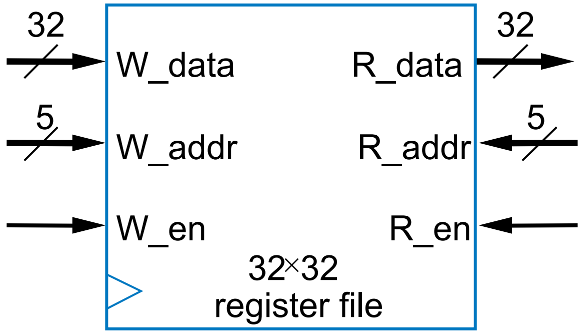




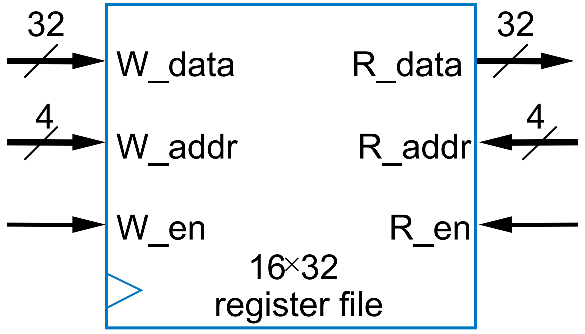
Register File -size



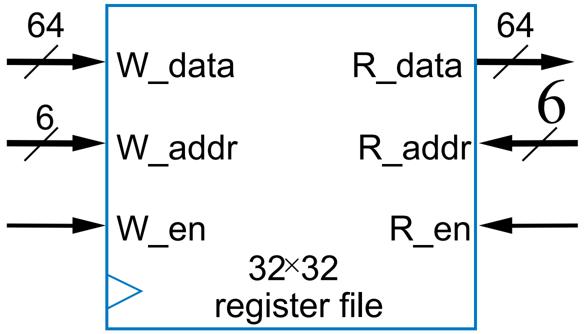
32x32



16x32

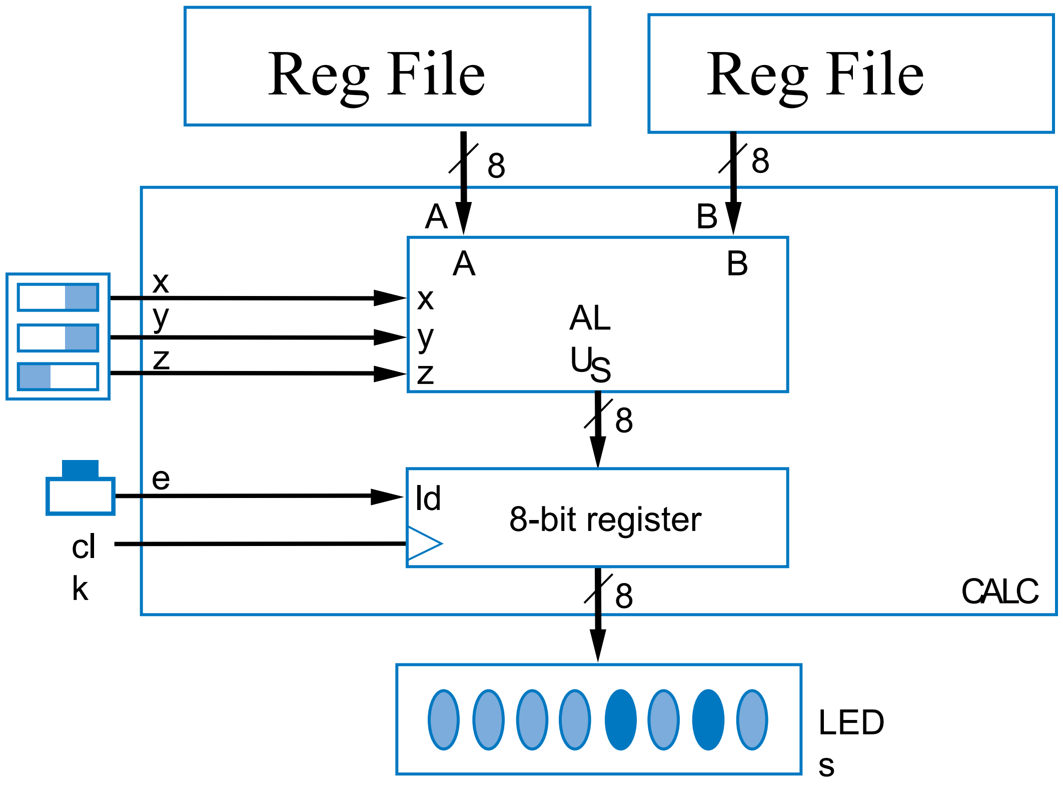


64x64

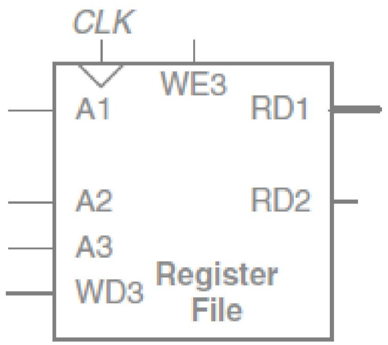


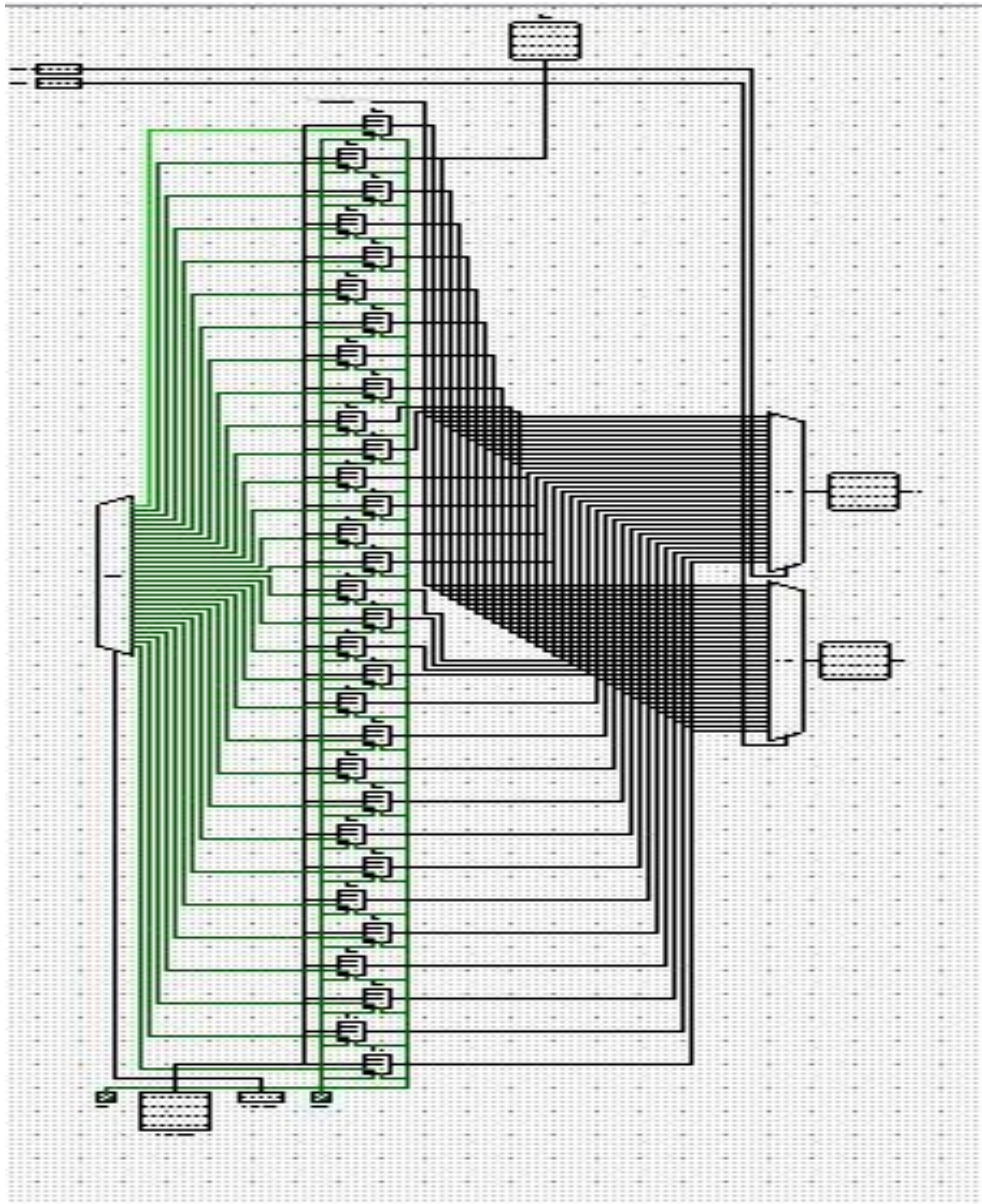
ALU Example:

- Design using ALU is elegant and efficient



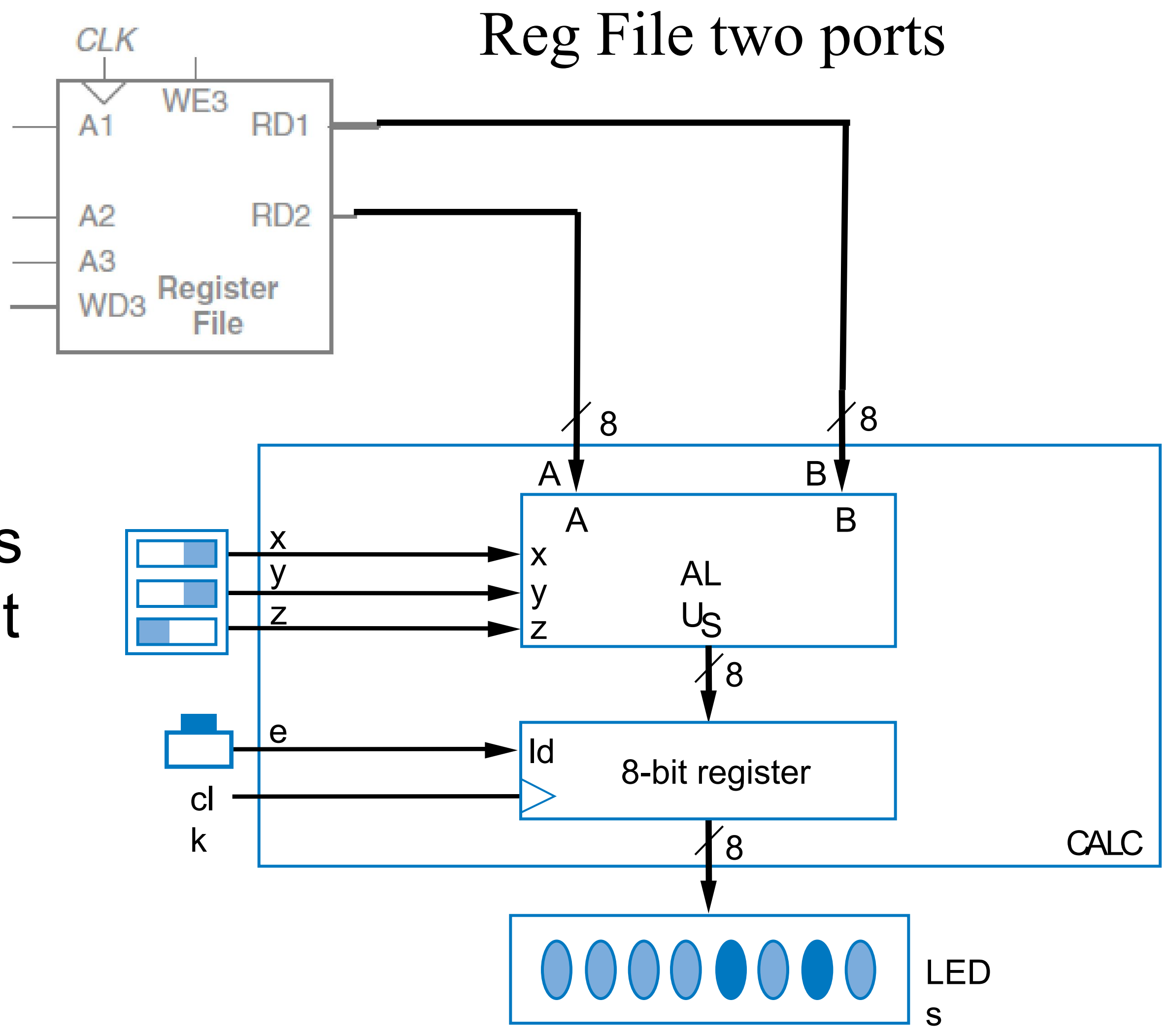
Register File with Two Ports





ALU Example:

- Design using ALU is elegant and efficient

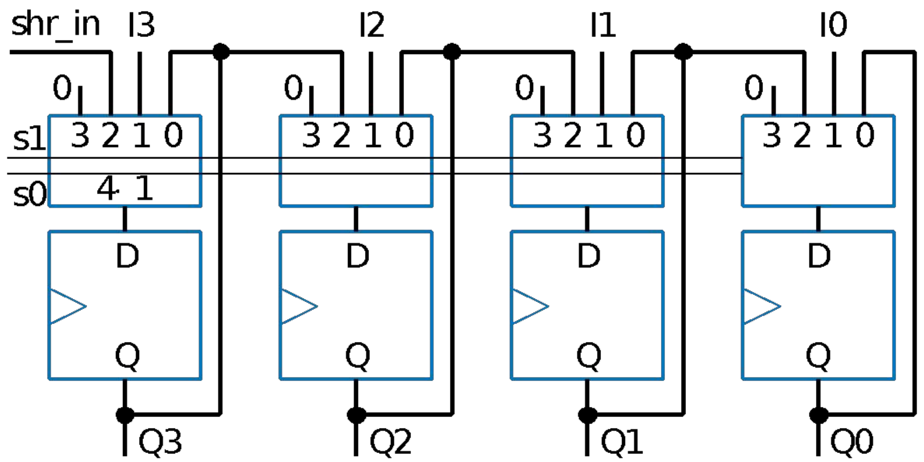


Multifunction Registers

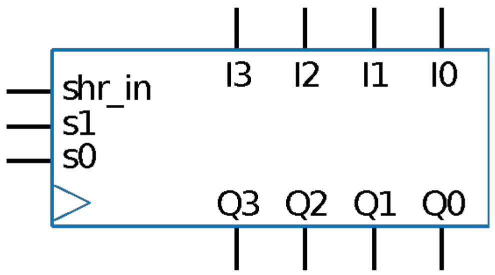
- Many registers have multiple functions
  - Load, shift, clear (load all 0s)
  - And retain present value, of course
- Easily designed using muxes
  - Just connect each mux input to achieve desired function

Functions:

s1	s0	Operation
0	0	Maintain present value
0	1	Parallel load
1	0	Shift right
1	1	(unused - let's load 0s)



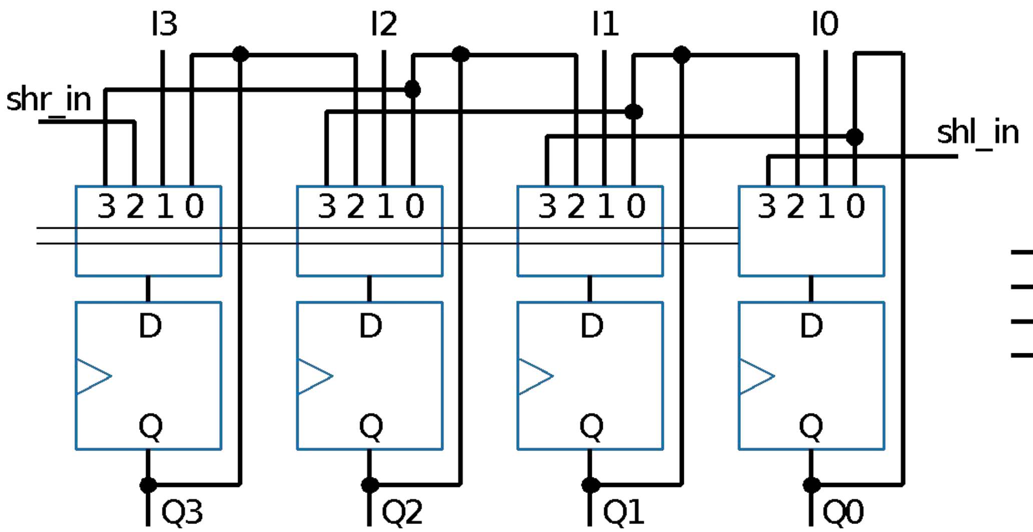
(a)



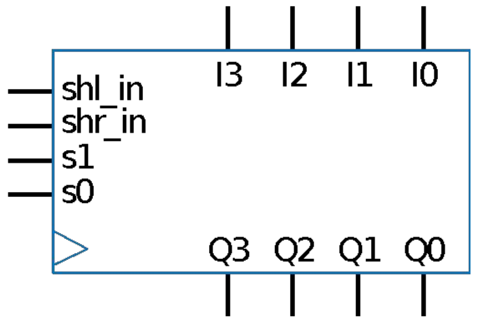
(b)

Multifunction Registers

s1	s0	Operation
0	0	Maintain present value
0	1	Parallel load
1	0	Shift right
1	1	Shift left



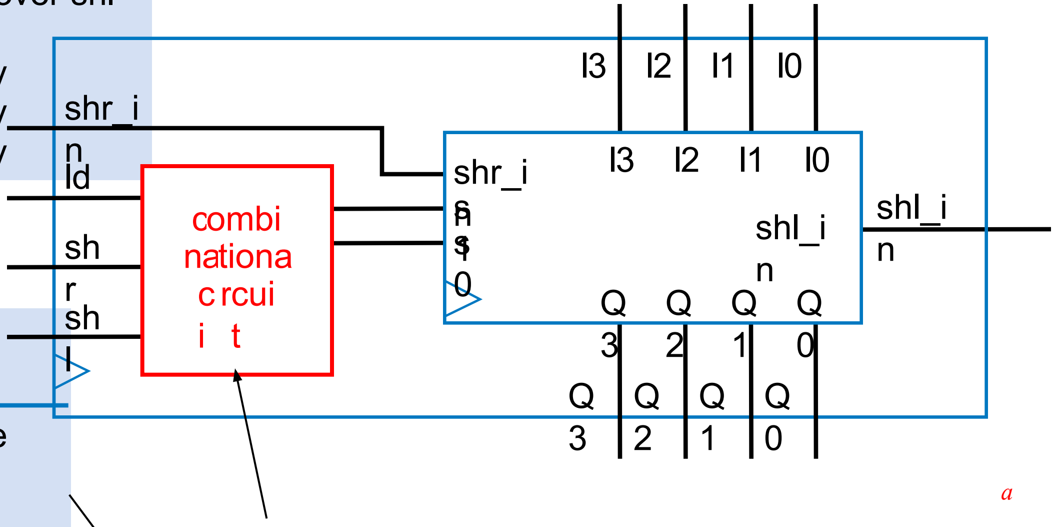
(a)



(b)

Multifunction Registers with Separate Control Inputs

ld	shr	shl	Operation
0	0	0	Maintain present value
0	0	1	Shift left
0	1	0	Shift right
0	1	1	Shift right – shr has priority over shl
1	0	0	Parallel load
1	0	1	Parallel load – ld has priority
1	1	0	Parallel load – ld has priority
1	1	1	Parallel load – ld has priority



Truth table for combinational circuit

Inputs			Outputs		Note
ld	shr	shl	s1	s0	Operation
0	0	0	0	0	Maintain value
0	0	1	1	1	Shift left
0	1	0	1	0	Shift right
0	1	1	1	0	Shift right
1	0	0	0	1	Parallel load
1	0	1	0	1	Parallel load
1	1	0	0	1	Parallel load
1	1	1	0	1	Parallel load

a

s1 = ld’*shr’*shl + ld’*shr*shl’ + ld’*shr*shl

s0 = ld’*shr’*shl + ld

a

a

Register Operation Table

- Register operations typically shown using compact version of table
  - X means same operation whether value is 0 or 1
    - One X expands to two rows
    - Two Xs expand to four rows
  - Put highest priority control input on left to make reduced table simple

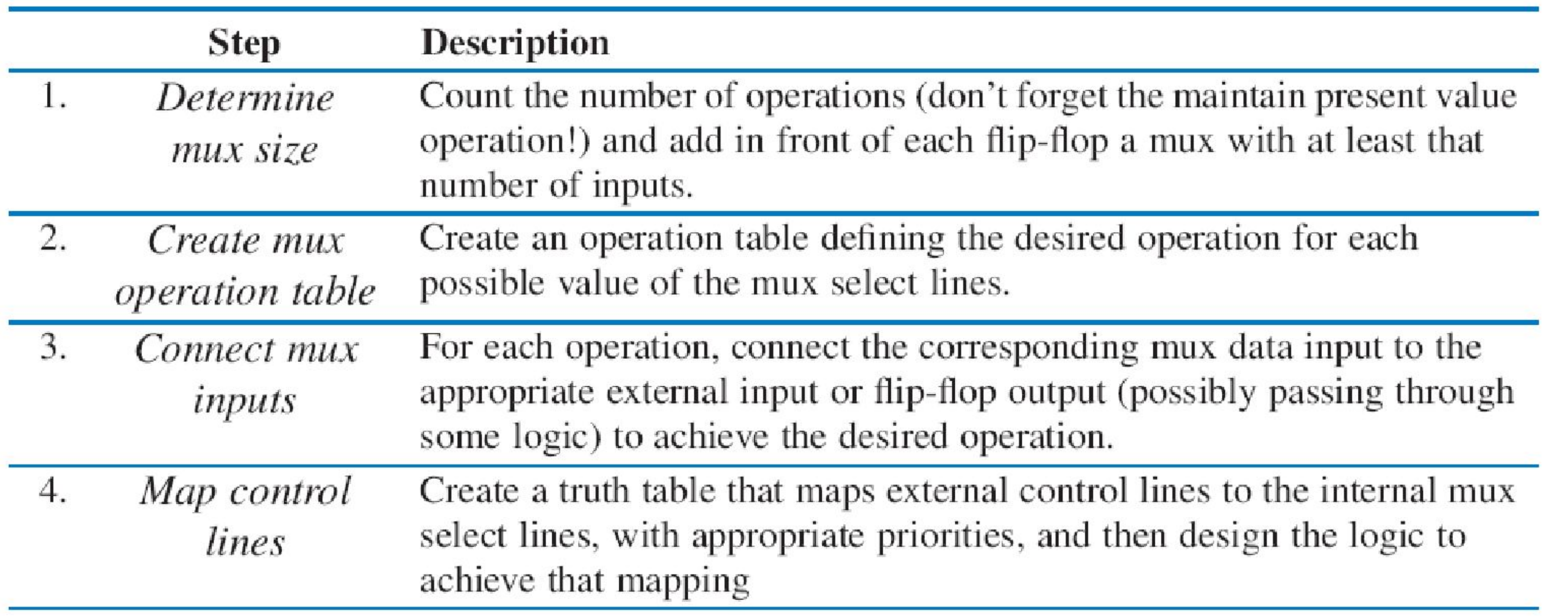
Inputs			Outputs		Note
ld	shr	shl	s1	s0	Operation
0	0	0	0	0	Maintain value
0	0	1	1	1	Shift left
0	1	0	1	0	Shift right
0	1	1	1	0	Shift right
1	0	0	0	1	Parallel load
1	0	1	0	1	Parallel load
1	1	0	0	1	Parallel load
1	1	1	0	1	Parallel load

ld	shr	shl	Operation
0	0	0	Maintain value
0	0	1	Shift left
0	1	X	Shift right
1	X	X	Parallel load

Register Design Process

- Can design register with desired operations using simple four-step process

TABLE 4.1 Four-step process for designing a multifunction register.



	Step	Description
1.	*Determine mux size*	Count the number of operations (don't forget the maintain present value operation!) and add in front of each flip-flop a mux with at least that number of inputs.
2.	*Create mux operation table*	Create an operation table defining the desired operation for each possible value of the mux select lines.
3.	*Connect mux inputs*	For each operation, connect the corresponding mux data input to the appropriate external input or flip-flop output (possibly passing through some logic) to achieve the desired operation.
4.	*Map control lines*	Create a truth table that maps external control lines to the internal mux select lines, with appropriate priorities, and then design the logic to achieve that mapping

Register Design Example

- Desired register operations
  - Load, shift left, synchronous clear, synchronous set

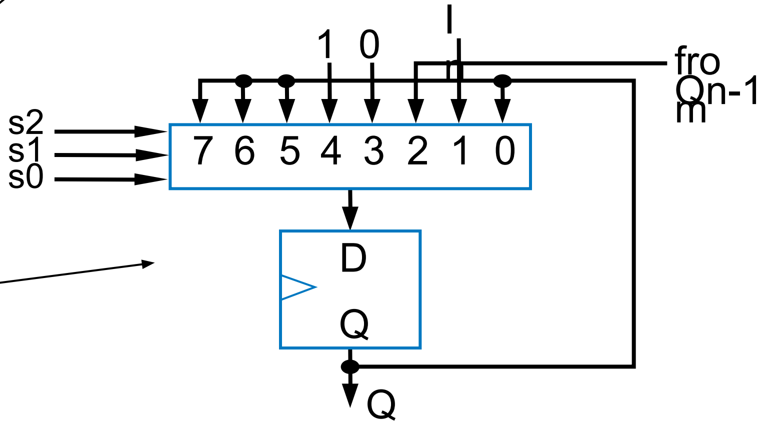
s2	s1	s0	Operation
0	0	0	Maintain present value
0	0	1	Parallel load
0	1	0	Shift left
0	1	1	Synchronous clear
1	0	0	Synchronous set
1	0	1	Maintain present value
1	1	0	Maintain present value
1	1	1	Maintain present value

<u>Step 1: Determine mux size</u>

a

5 operations: above, plus maintain present value (don't forget this one!)
--> **Use 8x1 mux**

<u>Step 2: Create mux operation table</u>



<u>Step 3: Connect mux inputs</u>

<u>Step 4: Map control lines</u>

s2 = clr'*set

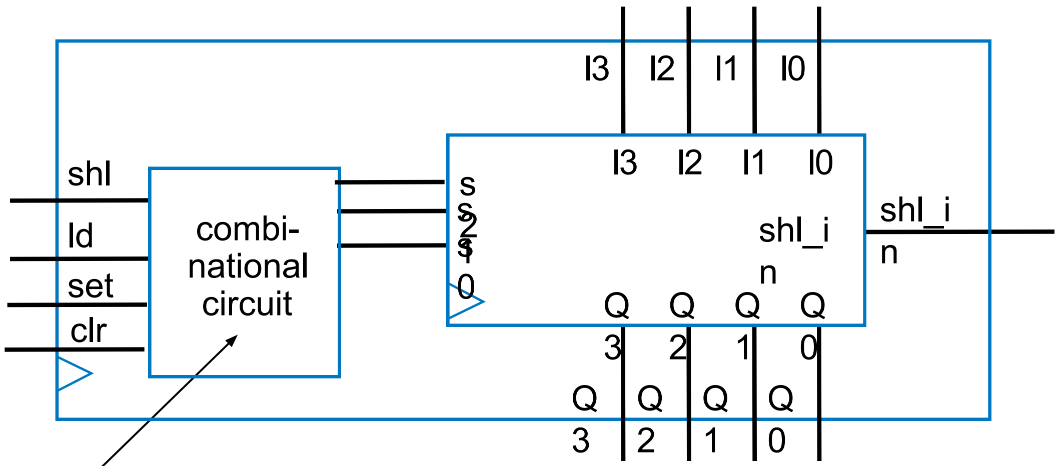
s1 = clr'*set'*ld'*shl + clr

s0 = clr'*set'*ld + clr

Inputs				Outputs			
clr	set	ld	shl	s2	s1	s0	Operation
0	0	0	0	0	0	0	Maintain present value
0	0	0	1	0	1	0	Shift left
0	0	1	X	0	0	1	Parallel load
0	1	X	X	1	0	0	Set to all 1s
1	X	X	X	0	1	1	Clear to all 0s

a

Register Design Example



Step 4: Map control lines

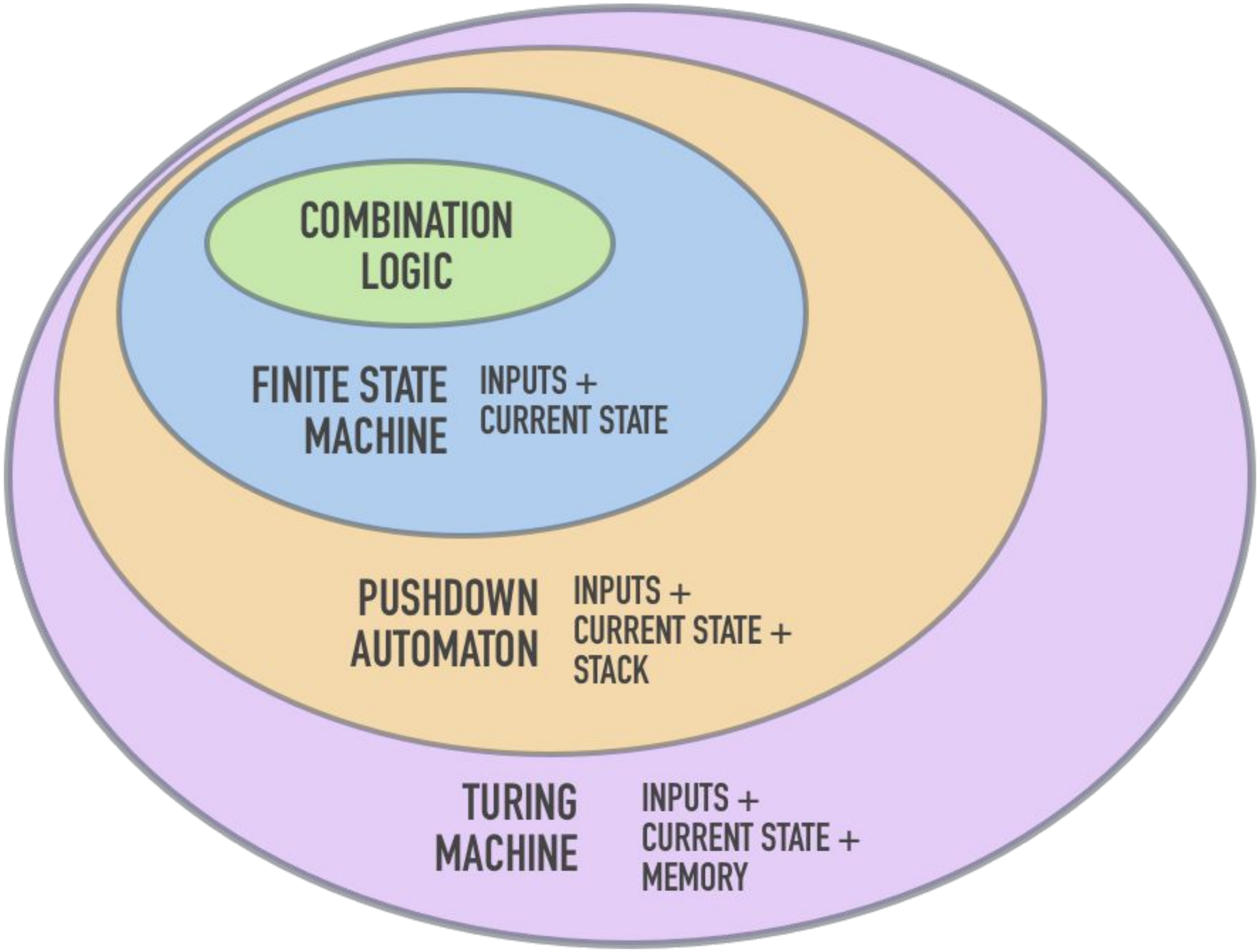
s2 = clr’*set

s1 = clr’*set’*ld’*shl + clr

s0 = clr’*set’*ld + clr

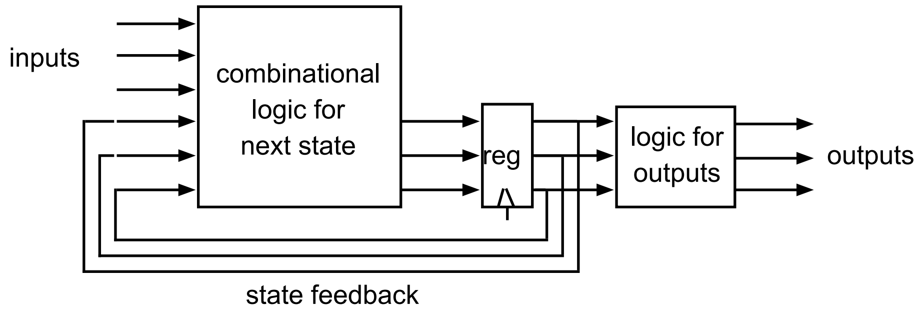
Inputs				Outputs			
clr	set	ld	shl	s2	s1	s0	Operation
0	0	0	0	0	0	0	Maintain present value
0	0	0	1	0	1	0	Shift left
0	0	1	X	0	0	1	Parallel load
0	1	X	X	1	0	0	Set to all 1s
1	X	X	X	0	1	1	Clear to all 0s

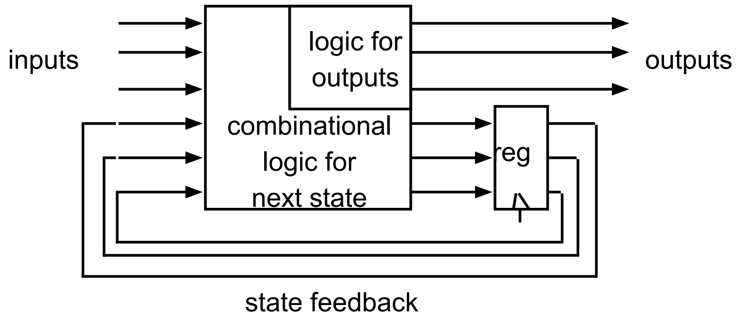
Finite-State Machines (FSMs) and Controllers


COMBINATION LOGIC
FINITE STATE MACHINE
INPUTS + CURRENT STATE
PUSHDOWN AUTOMATON
INPUTS + CURRENT STATE + STACK
TURING MACHINE
INPUTS + CURRENT STATE + MEMORY

Mealy and Moore machines

- Mealy machines tend to have fewer states
- Mealy machines react faster to inputs
- Moore machines are generally safer to use
  - outputs change at clock edge (always one cycle later)
  - in Mealy machines, input change can cause output change as soon as logic is done


inputs
combinational logic for next state
reg
logic for outputs
outputs
state feedback


inputs
logic for outputs
outputs
combinational logic for next state
reg
state feedback