Register File, Finite State Machines & Hardware Control Language

Avin R. Lebeck

Some slides based on those developed by Gershon Kedem, and by Randy Bryant and Dave O'Hallaron

Administrivia

- Homework #4 is up, due Oct 20
- Midterm: Median 90
- May be late for office hours Thursday Morning
- May move Monday afternoon office hours

Read

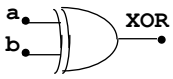
- Sections 4.1-4.3 of text
- Pragmatic Logic (PDF on blackboard or through Library)

Today

- Review Logic Design
  - Logisim demo (work through beginnergs guide)
- Memory: Latches, FlipFlops, Register file
- Finite State Machines: Sequential Circuits
- Hardware Control Language (if time)

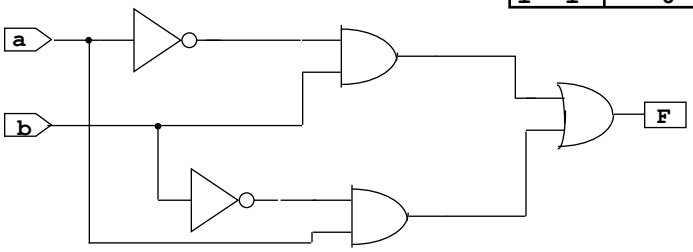
Review: Boolean Functions, Gates and Circuits

- **Circuits** are made from a network of gates. (function compositions).
- Logisim demo

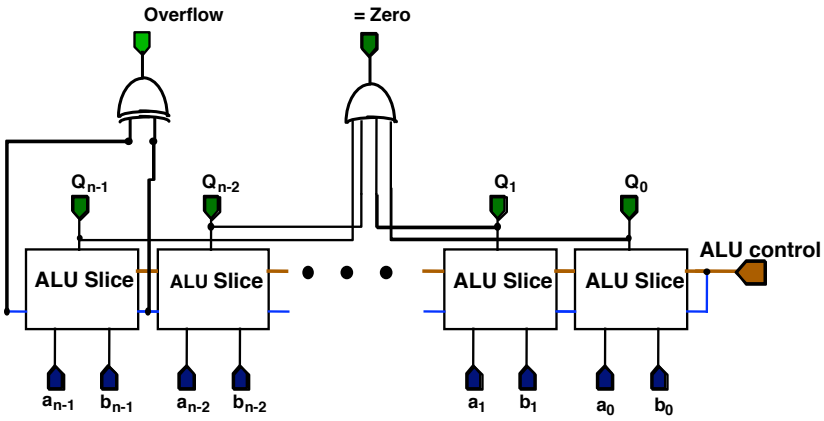


F = ~a*b + ~b*a

a	b	XOR(a,b)
0	0	0
0	1	1
1	0	1
1	1	0

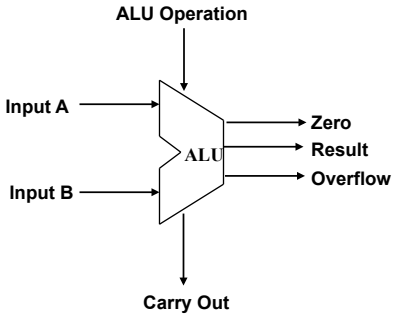


Review: The ALU



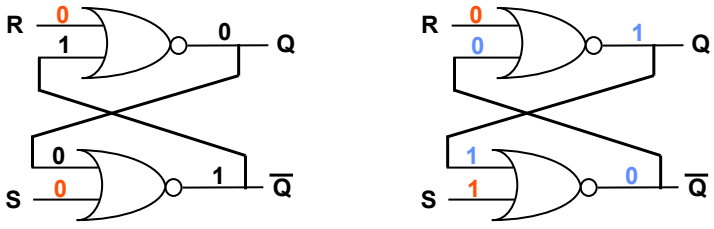
Review: Abstraction: The ALU

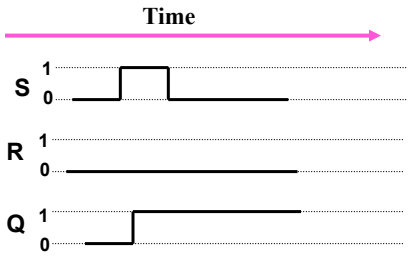
- General structure
- Two operand inputs
- Control inputs



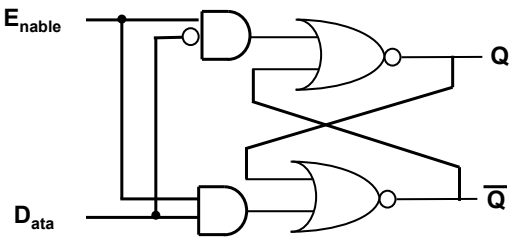
- We can build circuits for
  - Multiplication
  - Division
  - They are more complex

Review: Set-Reset Latch (Continued)

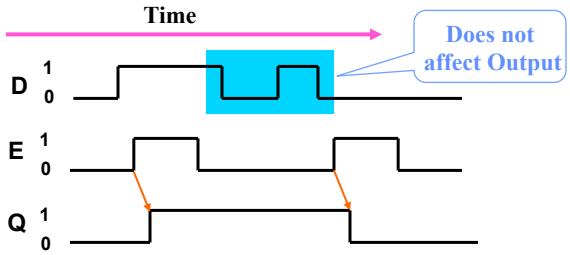




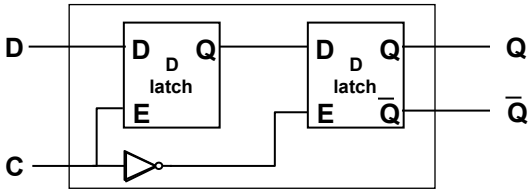
Data Latch (D Latch)



D	E	Q
0	1	0
1	1	1
-	0	Q



D Flip-Flop



- On C ↑ D is transferred to the first D latch and the second is stable.
- On C ↓ the output of the first stage is transferred to the second (output), and the first stage is stable.
- Key: Output changes only on the edge of a clock
- Logisim demos

Register File

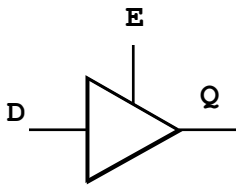
- Register File = the set of locations for register values
  - E.g., 32 32-bit registers
- How do I build a Register File using D Flip-Flops?
- What other components do I need?

Register File

- Circuit to determine which of 32 registers?
- Circuit to get just the data from one of 32 registers?

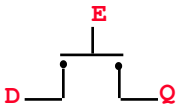
Tri-State Driver

- The Tri-State driver is like a (one directional) switch:
  - When the Enable is on (E=1) it transfers the input to the output.
  - When the Enable is off (E=0) it disconnects the output.



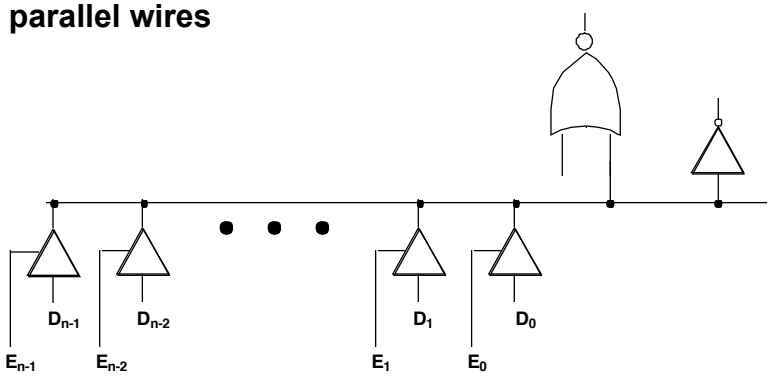
D	E	Q
0	1	0
1	1	1
–	0	Z

Z :- High Impedance

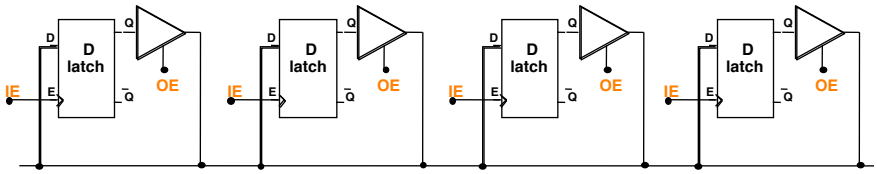


Bus Connections

- The Bus: Many to many connections.
- Mutual exclusion: At most one Enable is on!
- Control must ensure this!
- Note: Bus sometimes used to denote multiple parallel wires

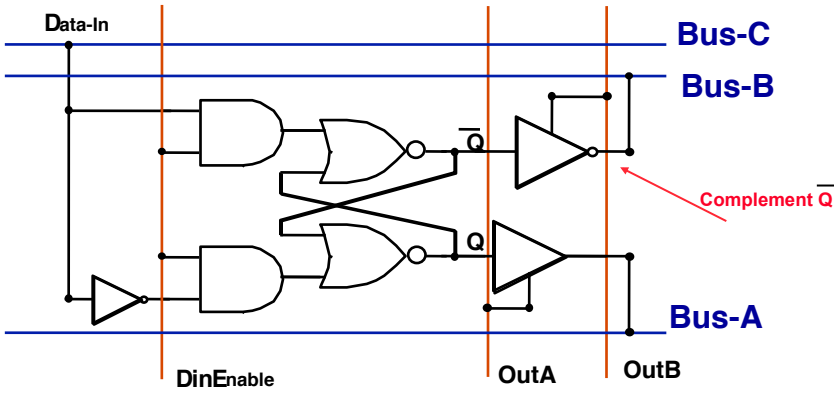


Register Cells on a bus



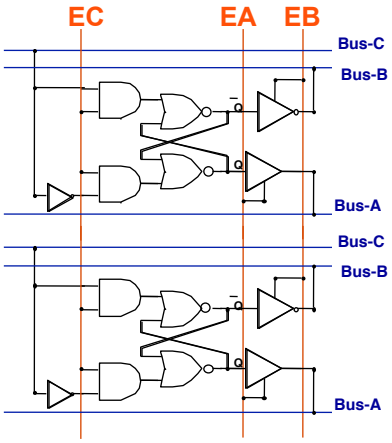
One can "source" and "sink" from any cell on the bus by activating the right controls,
IE--input enable, and OE--output enable.

3-Port Register Cell



- Stores one bit of a register
- Can Read onto Bus-A & Bus-B and Write from Bus-C Simultaneously

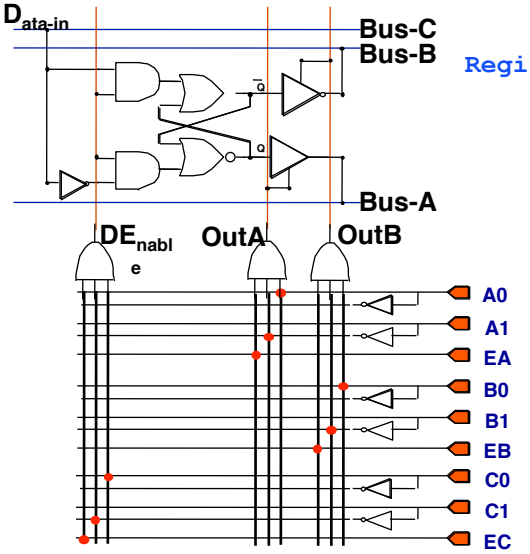
3-Port Register File



Bit-1

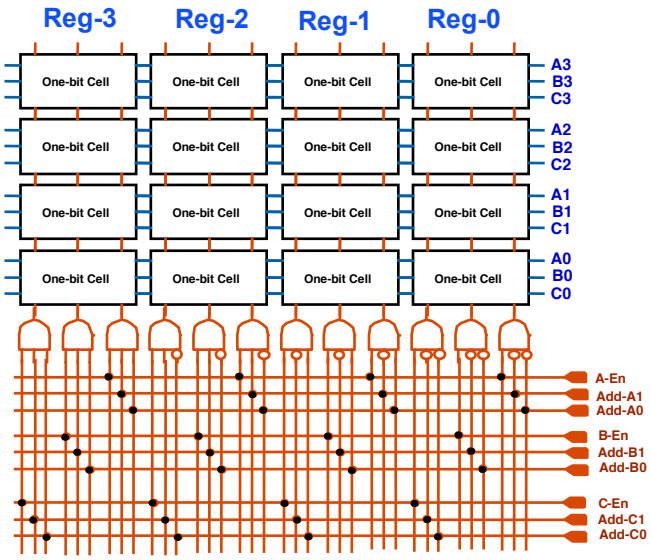
Bit-0

Address Decode Circuit



Register address: 01

Register File (Four 4-bit Registers)



Digital Logic Summary

- Given Boolean function, generate a circuit to "realize" the function.
- Constructed circuits that can add and subtract.
- The ALU: a circuit that can add, subtract, detect overflow, compare, and do bit-wise operations (AND, OR, NOT)
- Shifter
- Memory Elements: SR-Latch, D Latch, D Flip-Flop
- Tri-state drivers & Bus Communication vs. MUX
- Register Files
- Control Signals modify what circuit does with inputs
  - ALU, Shift, Register Read/Write
- Next
  - Finite State Machines
  - Hardware Control Language

Finite State Machine

- **S ={** $s_0, s_1, \ldots s_{n-1}\}$ is a finite set of states.
- **I =** $\{ i_0, i_1, \ldots i_{k-,1}\}$ is a finite set of input values.
- **O=** $\{ o_0, o_1, \ldots o_{m-1}\}$ is a finite set output values.

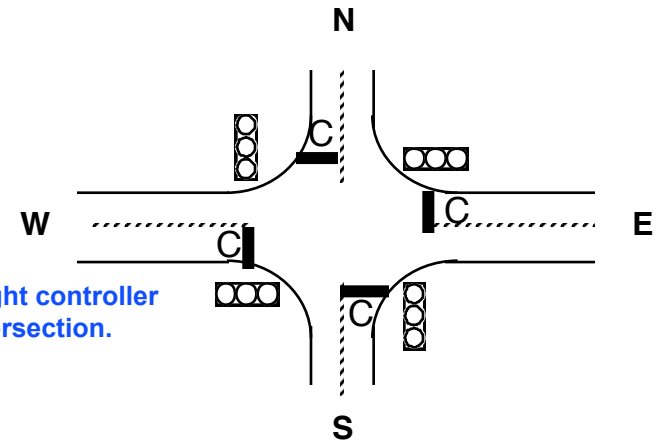
Definition: A finite state machine is a function F:(**S** x **I**)-> (**S** x **O**) that gets a sequence of input values $I_k \in I$, k = 0,1,2 , • • • and it produces a sequence of output values $O_k \in O$, k= 1,2, • • • such that:

$$\mathbf{F}(s_k, i_k) = (s_{k+1}, o_{k+1}) \quad K=0, 1, 2, \bullet\bullet\bullet$$

Finite State Machine
(Translation to English)

- **Finite State Machine is:**
  - **A machine with a finite number of possible states.**
  - **A machine with a finite number of possible Inputs.**
  - **A machine with a finite number of possible different outputs.**
  - **At each period (Clock cycle) the machine receives an input and it produces an output.**
  - **The output is a function of the machine input and current state.**
  - **After each period the machine changes state.**
  - **The new state is a function of the input and current state.**

Example: Traffic Light Controller



Traffic light controller at an intersection.

Finite State Machine (cont.)

- Example: Traffic lights controller:
  - There are four states:
    - NG: Green light in the north-south direction.
    - NY: Yellow light in the north-south direction.
    - EG: Green light at the East-West direction.
    - EY: Yellow light at the East-West direction.
  - There are four outputs:
    - (G;R): North-South green light, East-West red light
    - (Y;R): North-South yellow light, East West red light
    - (R;Y): North-South red light, East-West yellow light
    - (R;G): North-South red light, East-West green light
  - There are four inputs:
    - (c, c): Car at the North-South, Car at East-West
    - (c, nc) Car at North-South, No-car at East-West
    - (nc, c): No-car at North-South, Car at East-West
    - (nc, nc): No-car at North-South, No-car at East-West

FSM Example: Traffic Light

- State Transitions:

State	Input	Next-State	Output
NG	(-;NC)	NG	(G;R)
NG	(-;C)	NY	(G;R)
NY	-	EG	(Y;R)
EG	(NC;-)	EG	(R;G)
EG	(C;-)	EY	(R;G)
EY	-	NG	(R;Y)

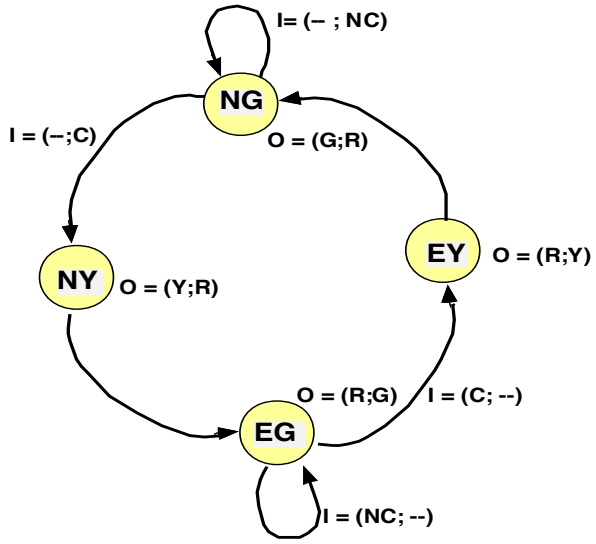
– means don't care

Format
(North/South; East/West)

Finite State Machine (cont.)

- Finite State Machines can be represented by a graph.
- The graph is called a State Diagram.
- The states are the nodes in the graph.
- The arcs in the graph represent state transitions.
- Each arc is labeled with the Inputs that cause the transition
- Nodes are labeled with the outputs.

FSM State Diagram
Example: Traffic light Controller



State Coding

State	Code
NG	00
NY	01
EG	10
EY	11

Enumerate States

Input	Code
(C;C)	11
(C;NC)	10
(NC;C)	01
(NC;NC)	00

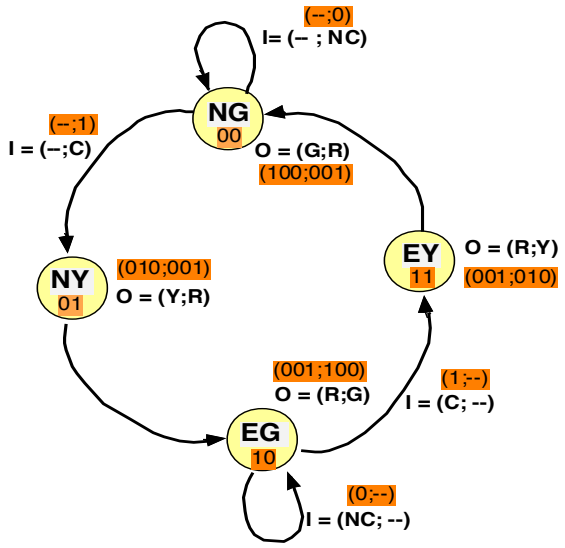
One bit for each Input
Input is either true or false

Output	Code
(R;G)	001100
(G;R)	100001
(Y;R)	010001
(R;Y)	001010

One bit per color for each light
GYRGYR

(North; East)

Coded State Diagram



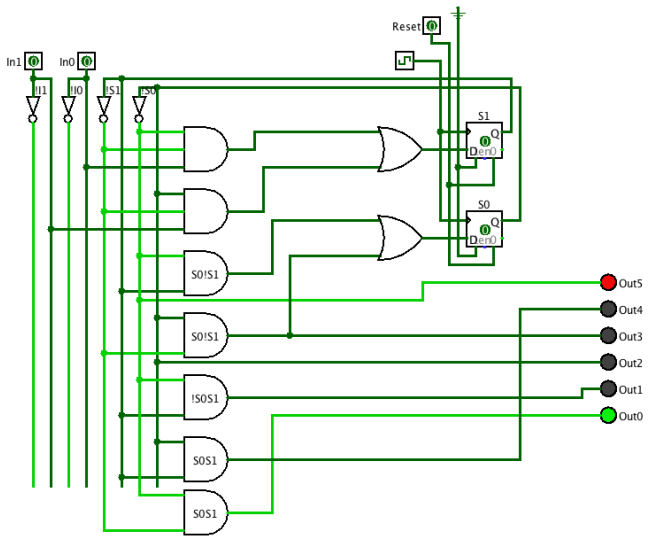
Example: Traffic Light Controller

S = State, bits are S0 and S1
NS = Next State, bits are NS0 and NS1

IN 01	S 01	NS 01	OUT 012345
0-	00	00	100001
1-	00	01	100001
--	01	10	010001
-0	10	10	001100
-1	10	11	001100
--	11	00	001010

```
NS1 = S0'*S1'*I0+S0*S1'*I1
    = S1'*(S0'I0+S0*I1)
NS0 = S0'*S1+S0*S1'*I1'+S0*S1'*I1
    = S0'*S1+S0*S1'
OUT0 = S0'*S1'
OUT1 = S0'*S1
OUT2 = S0*S1'+S0*S1= S0
OUT3 = S0*S1'
OUT4 = S0*S1
OUT5 = S0'*S1'+S0'*S1= S0'
```

Traffic Controller FSM implementation



General Method for FSM design

- **Determine the problem:**
  1. **Draw the state diagram,**
  2. **Write the truth table,**
  3. **Write sum-of-products equations**

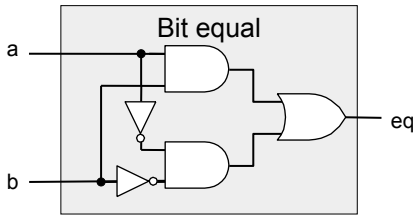
A Simple Arrow FSM

- Consider those flashing arrow signs
- No light, one light, two lights, three lights
  - > >> >>>
- Let's design the FSM to control this sign

Pattern Recognizer

- A pattern recognizer examines a sequence of inputs to detect when it sees the pattern 101. When it sees this pattern its output is 1 forever.
- Let's design the FSM

Bit Equality



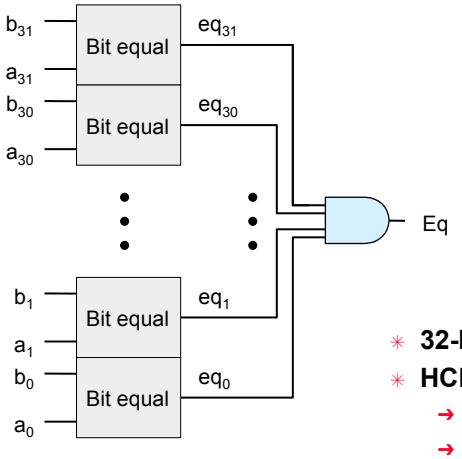
HCL Expression

```
bool eq = (a&&b)||(!a&&!b)
```

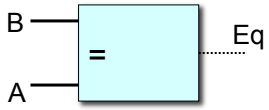
- Generate 1 if a and b are equal

- Hardware Control Language (HCL)
  - Very simple hardware description language
    - Boolean operations have syntax similar to C logical operations
  - We'll use it to describe control logic for processors

Word Equality



Word-Level Representation

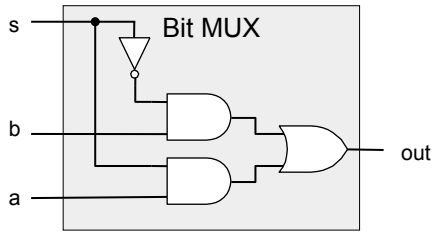


HCL Representation

```
bool Eq = (A == B)
```

- 32-bit word size
- HCL representation
  - Equality operation
  - Generates Boolean value

Bit-Level Multiplexor

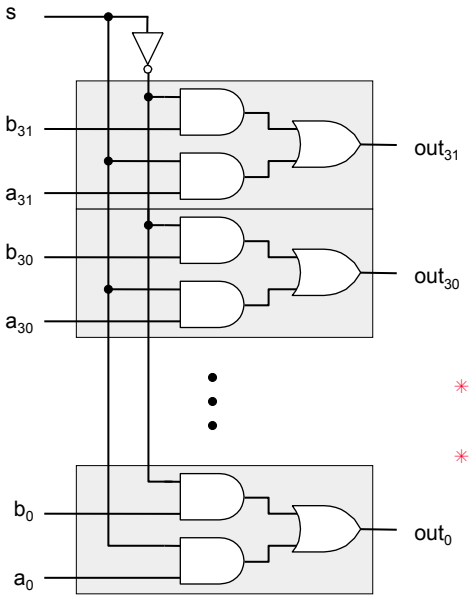


HCL Expression

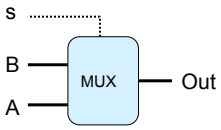
```
bool out = (s&&a)||(!s&&b)
```

- Control signal s
- Data signals a and b
- Output a when s=1, b when s=0

Word Multiplexor



Word-Level Representation



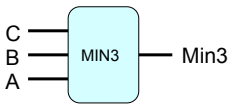
HCL Representation

```
int Out = [
  s : A;
  1 : B;
];
```

- Select input word A or B depending on control signal s
- HCL representation
  - → Case expression
  - → Series of test : value pairs
  - → Output value for first successful test

HCL Word-Level Examples

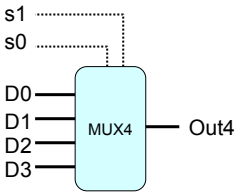
Minimum of 3 Words




```
int Min3 = [
  A < B && A < C : A;
  B < A && B < C : B;
  1              : C;
];
```

- Find minimum of three input words
- HCL case expression
- Final case guarantees match

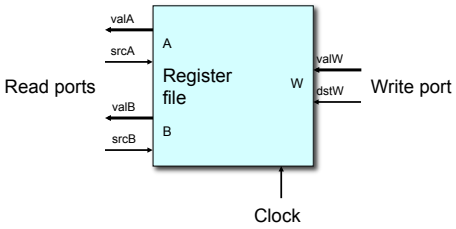
4-Way Multiplexor




```
int Out4 = [
  !s1&&!s0: D0;
  !s1     : D1;
  !s0     : D2;
  1       : D3;
];
```

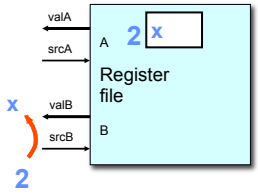
- Select one of 4 inputs based on two control bits
- HCL case expression
- Simplify tests by assuming sequential matching

Random-Access Memory

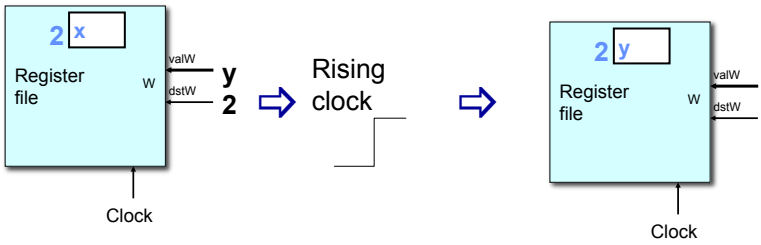


- **Stores multiple words of memory**
  - Address input specifies which word to read or write
- **Register file**
  - Holds values of program registers
  - `%eax`, `%esp`, etc.
  - Register identifier serves as address
    - ID 8 implies no read or write performed
- **Multiple Ports**
  - Can read and/or write multiple words in one cycle
    - Each has separate address and data input/output

Register File Timing



- **Reading**
  - Like combinational logic
  - Output data generated based on input address
    - ➔ After some delay
- **Writing**
  - Like register
  - Update only as clock rises



Summary

Finite State Machines

- Inputs, Current State
- Compute Outputs and Next State

HCL

- Language to express boolean logic
- Also, word level functions

- Homework #4