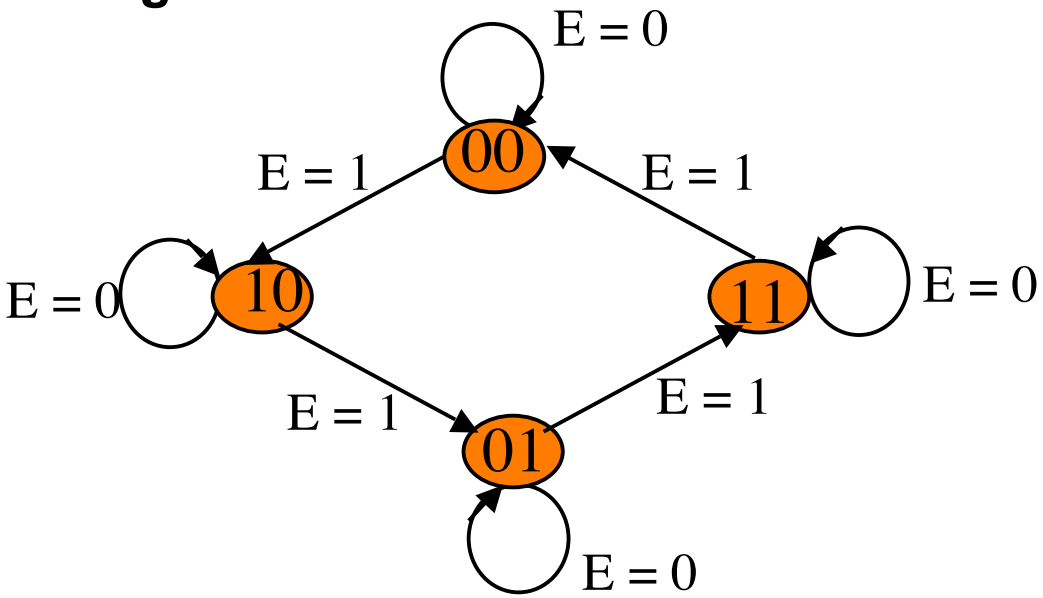
Design of Sequential Circuits

- Seven Steps:
  - Construct a **state diagram** (showing contents of flip flop and inputs with next state)
  - Assign **letter variables** to each flip flop and each input and output variable
  - Construct a **state table** (m flip flops, n inputs, p outputs give 2^{n+m} rows, and n + p + 2*m columns!)
  - Choose a **flip flop type**
  - Extend state table into an **excitation table** (given current state and next state, what are flip flop inputs?)
  - Obtain **Boolean equations** for flip flop inputs and any outputs of the combinational part of the circuit
  - **Draw** the circuit!

Synchronous Binary Counter with Enable

- Informal description: the counter counts 00,10,01,11 when E (enable) = 1. When E = 0 nothing happens.
- **Step 1: State diagram**



- Inside the states are the flip flop values, and the transitions are labelled with input values (E). There are no outputs.

Synchronous Binary Counter

- **Step 2: Assign names A and B to flip flops.**
- **Step 3: State table**

Present state		input	next state	
A	B	E	A	B
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Excitation Tables

- Characteristic tables of flip flops give next state from current inputs and state. During design we know next state and current state, so need to know what inputs to apply.
- This is given by the **excitation table.** E.g.
- **JK flip flop** excitation table is

Q(t)	Q(t+1)	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

x means “don't care”.
Eg 00 means no change
01 means clear
Both have the same effect.

Other Excitation Tables

- **SR flip flop**

Q(t)	Q(t+1)	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

- **D flip flop**

Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

Synchronous Binary Counter

- **Step 4: Choose JK flip flops**
- **Step 5: Excitation table**

present state		input	next state		Flip Flop inputs			
A	B	E	A	B	J_A	K_A	J_B	K_B
0	0	0						
0	0	1						
0	1	0						
0	1	1						
1	0	0						
1	0	1						
1	1	0						
1	1	1						

- The clock is implicit in the above.

Synchronous Binary Counter

- **Step 6: Derive Boolean Equations**
  - Inputs to the circuit are E, A, B.
  - Outputs from the circuit are J_A, K_A, J_B, K_B. (and A and B)
  - Use the state table as a truth table to derive Karnaugh maps and Boolean functions.
- Result:
  - $J_A = E$
  - $K_A = E$
  - $J_B = A.E$
  - $K_B = A.E$

Synchronous Binary Counter

- **Step 7: Draw the circuit!**

Digital Components

- Sequential and Combinational systems can be built from basic gates (AND, OR, NOT) and flip-flops
- Other (slightly) larger scale components exist
  - to make design easier

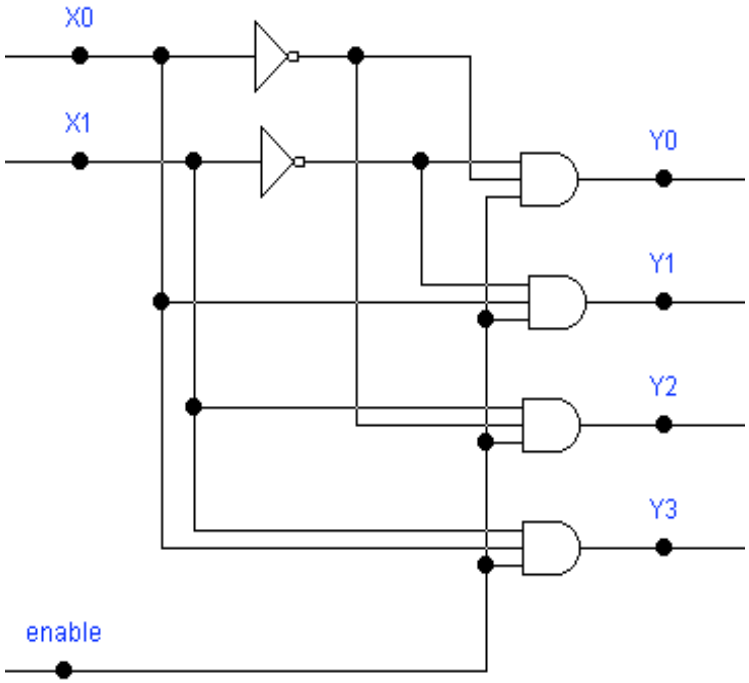
Decoder

- A **decoder**
  - takes a binary number represented in n bits (so 0 - (2^n - 1) values possible)
  - produces output on the appropriate one of m (= 2^n) output lines
- These are also called *n* to *m* line decoders.
- Applications of a decoder include:
  - detecting specific inputs
  - selecting chips
  - selecting I/O ports
  - comparing numbers
  - converting from one base to another
  - ….

Decoder

- Eg 2 to 4 line decoder

X1	X0	out
0	0	$Y_0=1$
0	1	$Y_1=1$
1	0	$Y_2=1$
1	1	$Y_3=1$

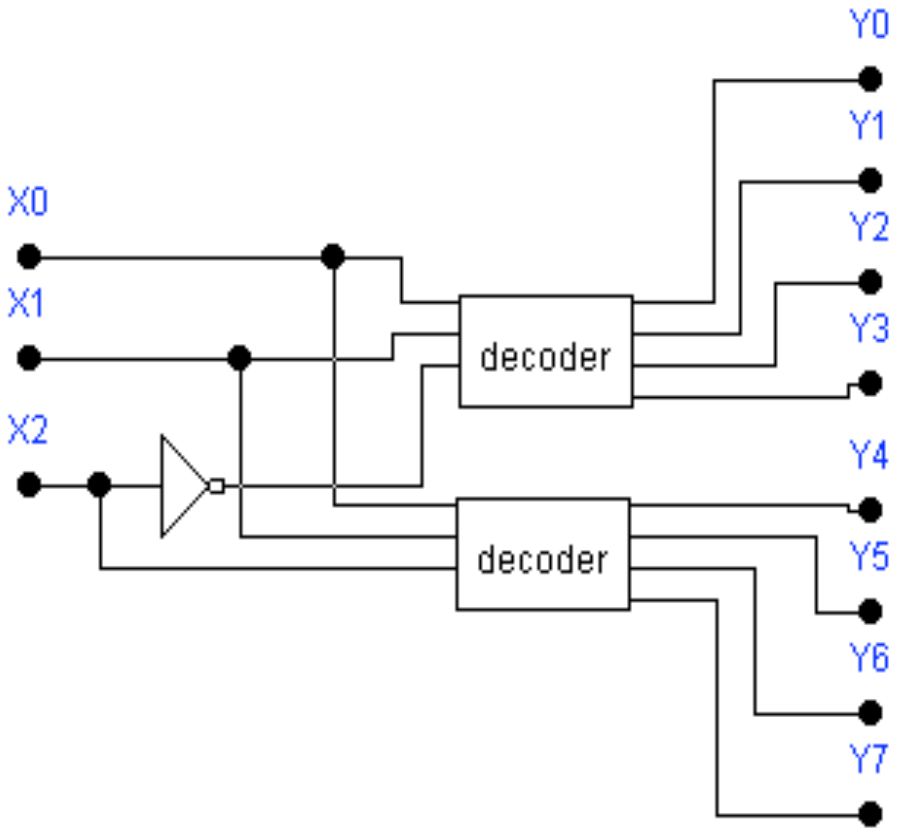


- all other outputs = 0
- Enable input means all outputs 0 if Enable = 0

Decoder Expansion

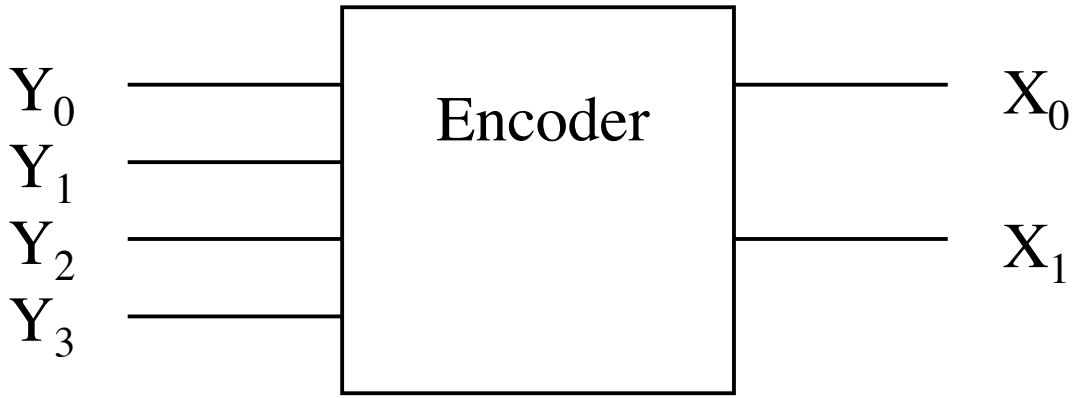
- Enable is useful if you need to join two decoders together.
- Eg

- When X_2 = 0 the bottom decoder (MSB) is disabled
  - X_0 and X_1 produce data outputs Y_0 - Y_3.
- When X_2 = 1 the top decoder is disabled
  - X_0 and X_1 produce data outputs Y_4 - Y_7



Encoder

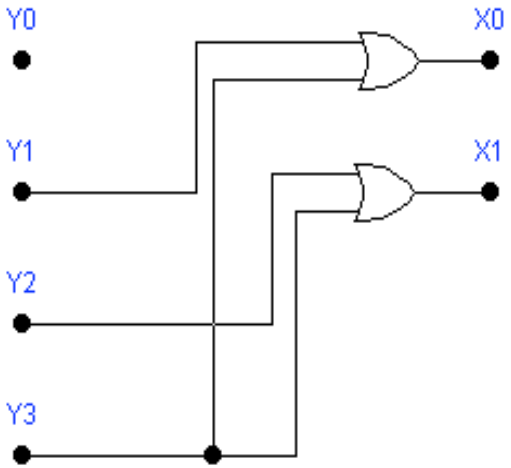
- An **Encoder** performs the inverse operation to a decoder.
  - Given m = 2^n inputs
    - outputs the corresponding binary code on n lines
- Applications:
  - keyboard encoder - one key press makes an input line high
  - converting numbers from one base to another



Encoder

- OR gates can be used to implement the encoder
- Only 1 input should be 1 at any time

$$X_0 = Y_1 + Y_3$$
$$X_1 = Y_2 + Y_3$$



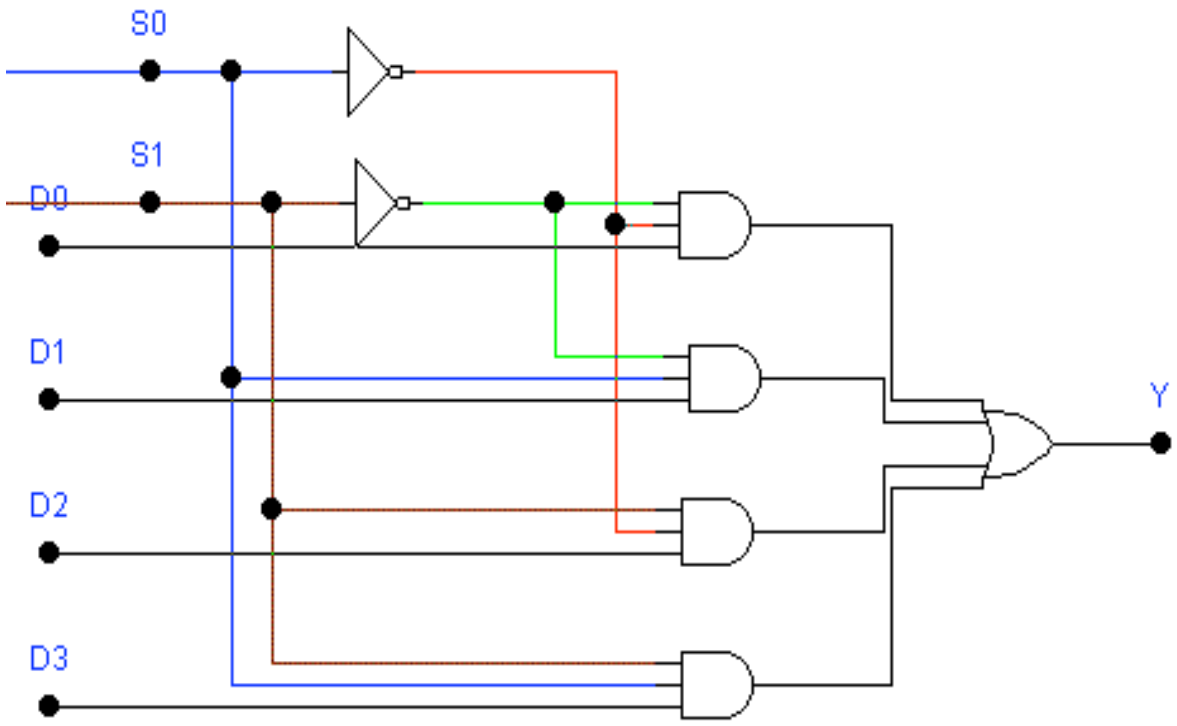
Multiplexers

- A **multiplexer** is a data selector.
  - Data is received on one of 2^n input lines
  - and directed to a single output line.
- The selection of a particular input line is determined by the selection inputs.
- A 2^n to 1 multiplexer has 2^n data input lines, n select lines and 1 output.
- Applications:
  - parallel to serial conversion
  - hard coding of functions (see later)

Multiplexer

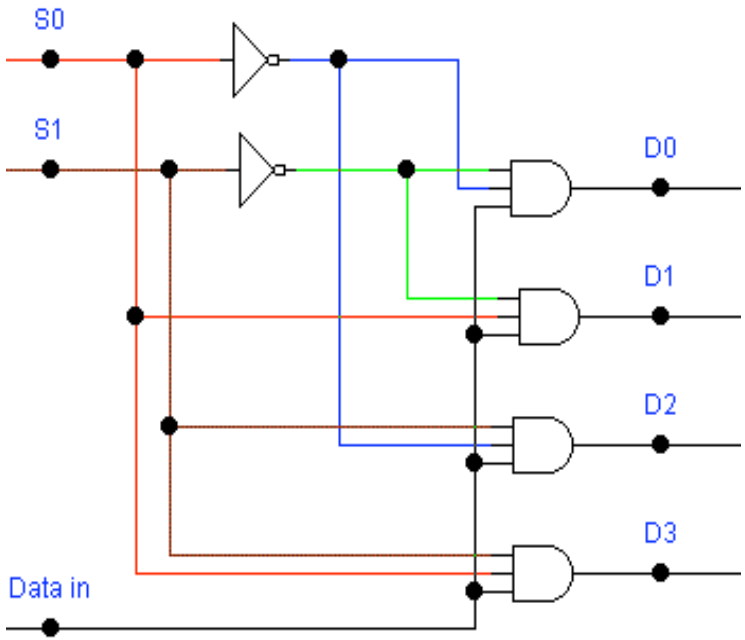
- Eg 4 to 1 multiplexer
- Whichever input S_1 S_0 selects is piped straight to Y.

S_1	S_0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3



Demultiplexer

- The converse operation to the Multiplexer
- Application - Data distribution



Register Transfer and MicroOperations

- Basic gates and larger components can be put together with data and control paths to construct more complex systems
- How do we talk about the dynamic changes in systems, i.e.
  - A set of registers and their purpose
  - The operations which may be performed on the content of the registers
  - The controls which determine what happens and when
- Operations on data stored in registers are called micro operations: logical, shift and arithmetic operations
- The **register transfer language** is a concise and precise means of describing those operations

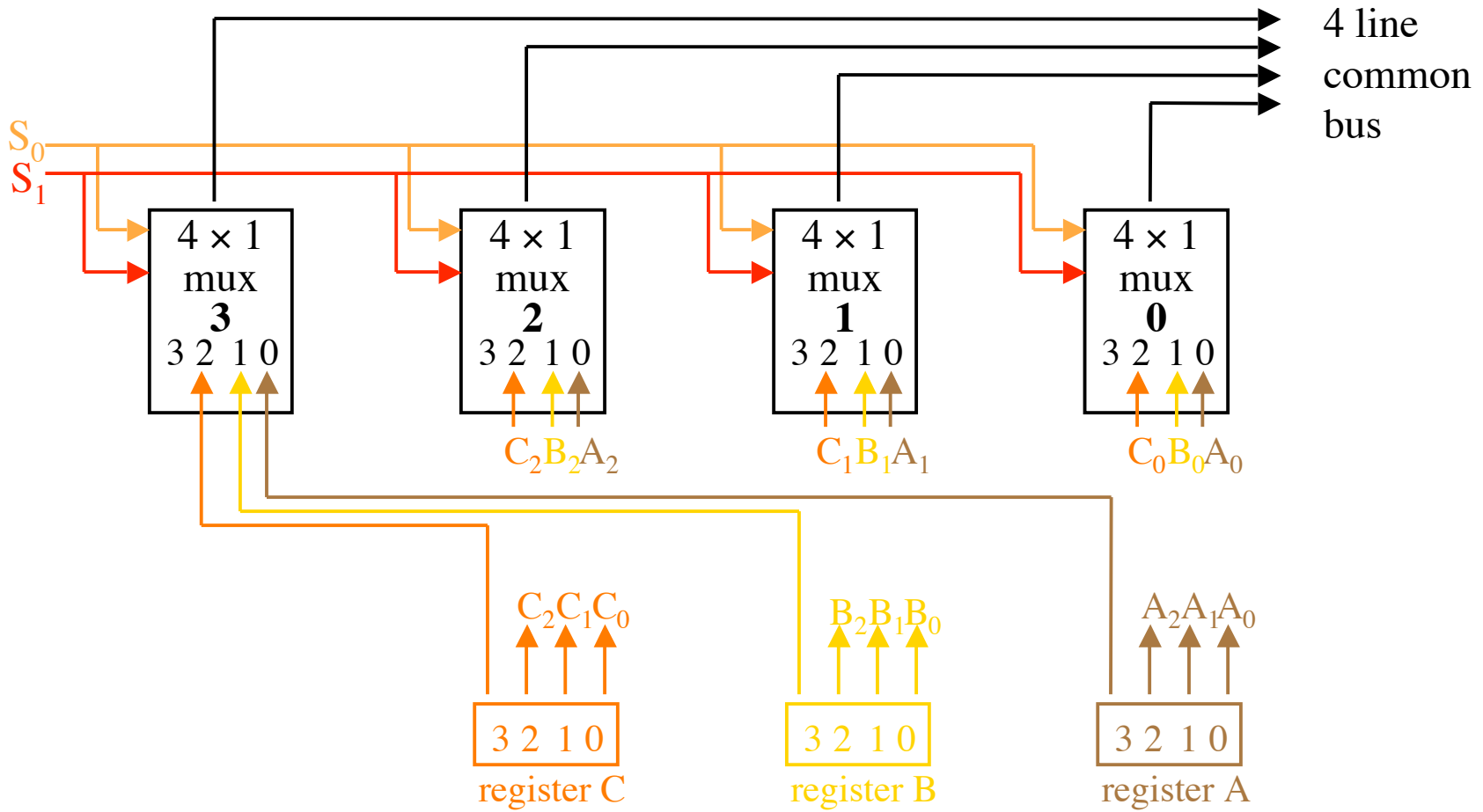
Bus construction

- How does information get from one register to another?
- Direct connection of all registers
  - impossible - there are too many wires
- Common bus system
  - a set of common lines - one for each bit - through which the binary information is transferred
  - Control signals determine which register are active at any point
    - and therefore which data is present on the bus
    - or which register should be loading the data from the bus

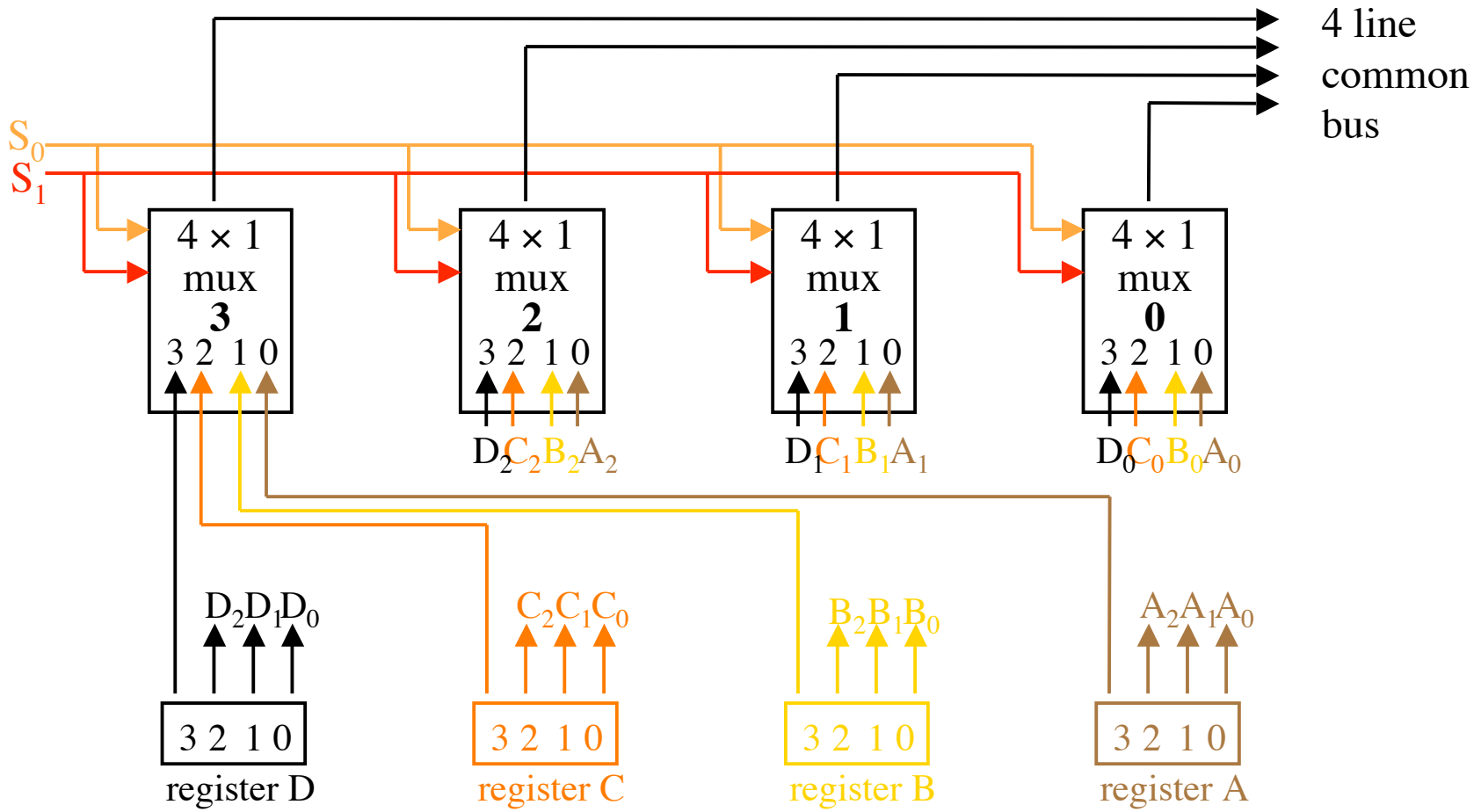
Bus construction (2)

- Use multiplexers to channel the data
- Selection lines choose which register delivers information to the bus
- The number of multiplexers required is *n*, that is, no. of bits in each register
- Each multiplexer is a **k to1 mux**, where *k* is the number of registers
- The bus is usually implicit in the register transfer language constructs

Bus Construction (3)

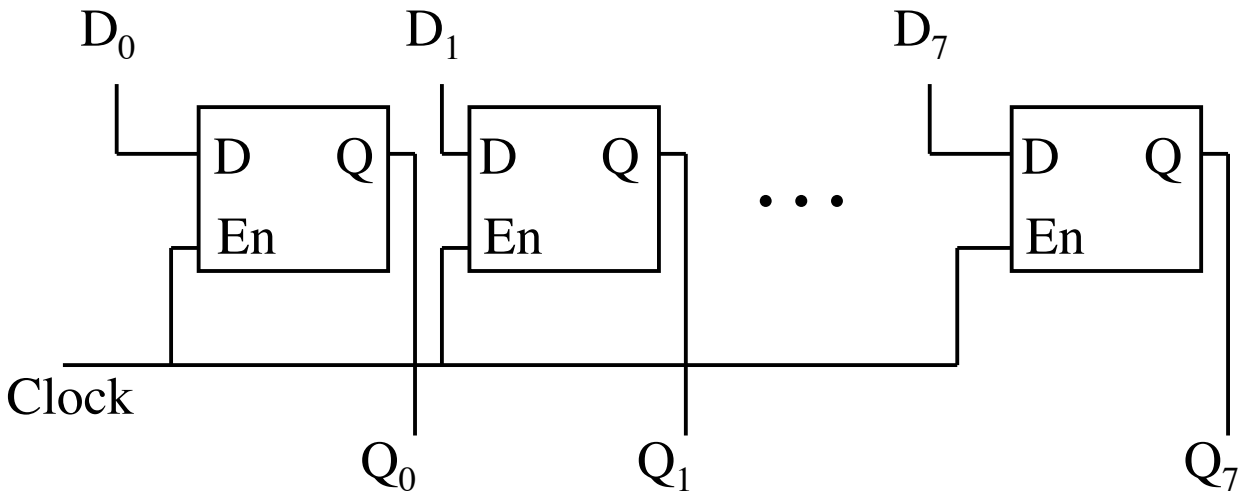


Buses (4): Another example



Registers

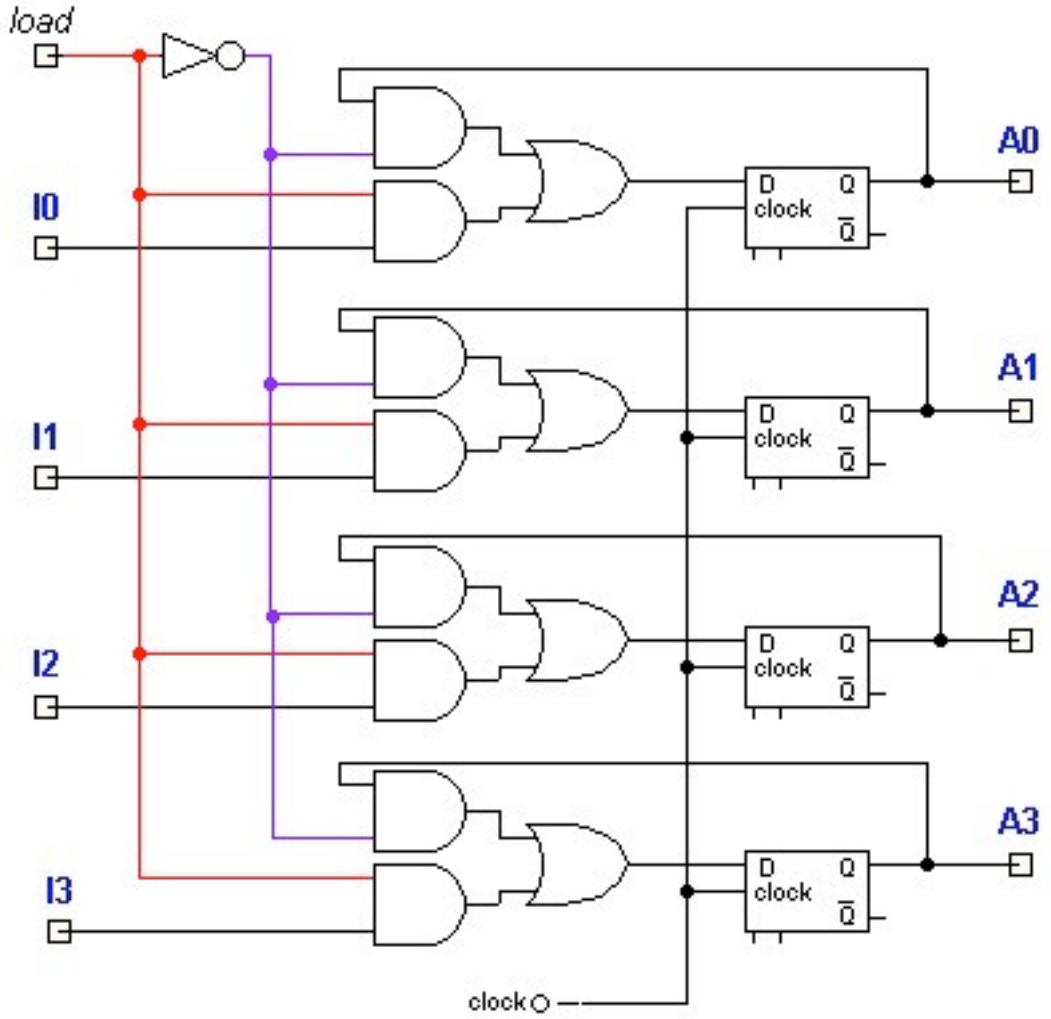
- Simple register constructed from D type flip flops.



- 8 bit register with parallel load. This uses the clock as an enable. Not always good!

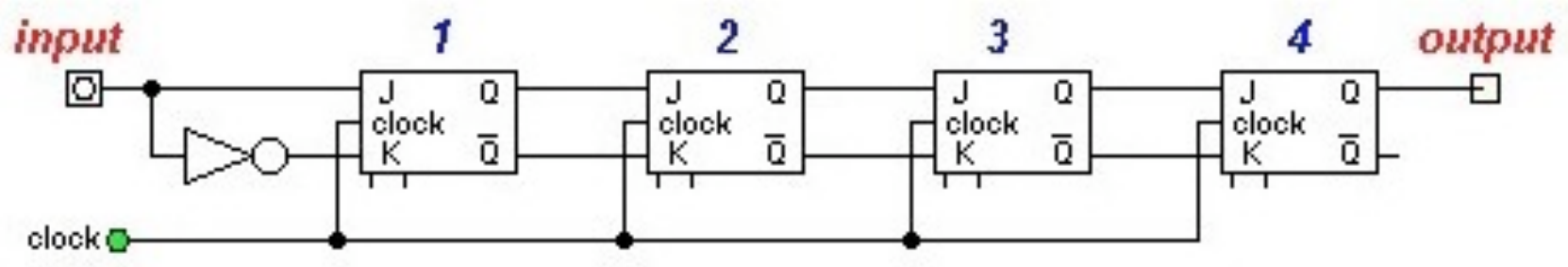
Synchronised Registers with load

- Here, the clock provides pulses (to synchronise the whole system).
- The enable (load) signals when data (I0 to I3) is to be loaded.
- If no load signal then old values (A0 to A3) are fed back.



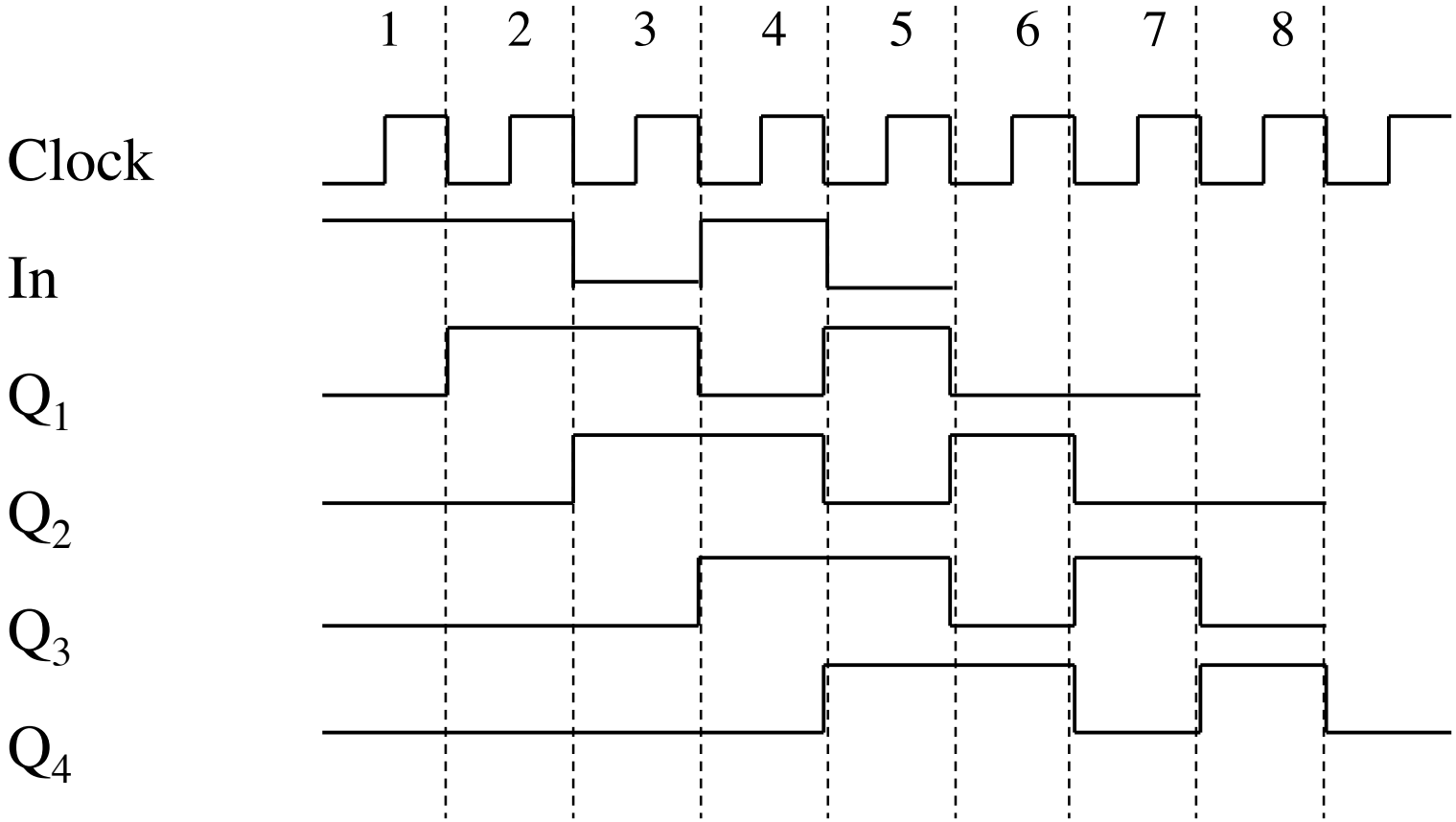
Shift Register

- Can do more than just store data - can also manipulate it.
  - shift its contents in one or both directions
- Implemented by a chain of flip flops.



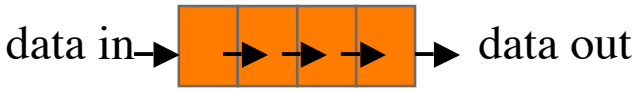
- serial in/serial out - leftmost is least significant bit.
- data shifts left to right, moving from one flip flop to the next every clock pulse - recall that this implements multiplication by 2 (assuming LSB is at the left)

Timing Diagram

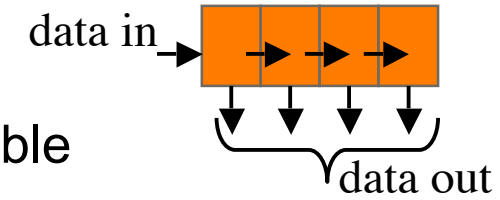


Example: Universal Shift Register

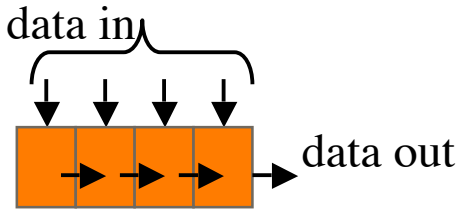
- Can do serial in/serial out (SISO)



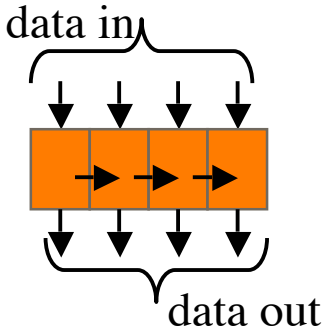
- serial in/parallel out (SIPO)
  - all Q outputs are externally accessible



- parallel in/serial out (PISO)
  - all P inputs simultaneously



- parallel in/parallel out (PIPO)



Bidirectional shift register with parallel load

- Has everything! (a Universal Register)
  - Clock input
  - Shift right and serial line to take new LSB
  - Shift left and serial line to take new MSB
  - Parallel load with n lines
  - Parallel output on n lines
  - Control state
- Applications:
  - time delay (input pulse advances each clock tick)
  - conversion of parallel data to serial data and back again
  - counter (seen already)
  - arithmetic (multiplication and division)

Bidirectional Shift Register

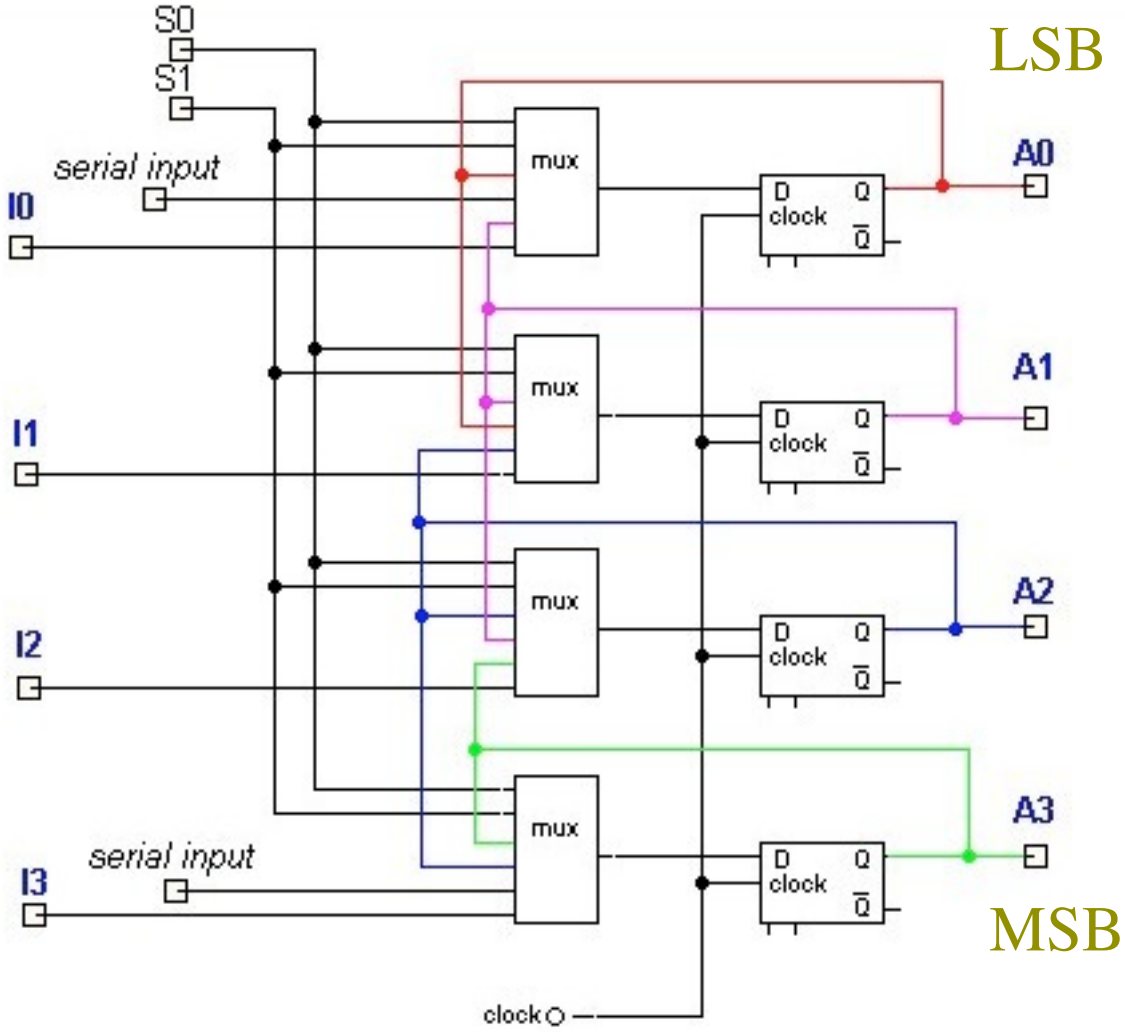
S_1 and S_0 select data inputs

S_1 S_0 = 00 means no change

S_1 S_0 = 01 means shift right (multiply: serial input for A_0 previous flip-flop for others)

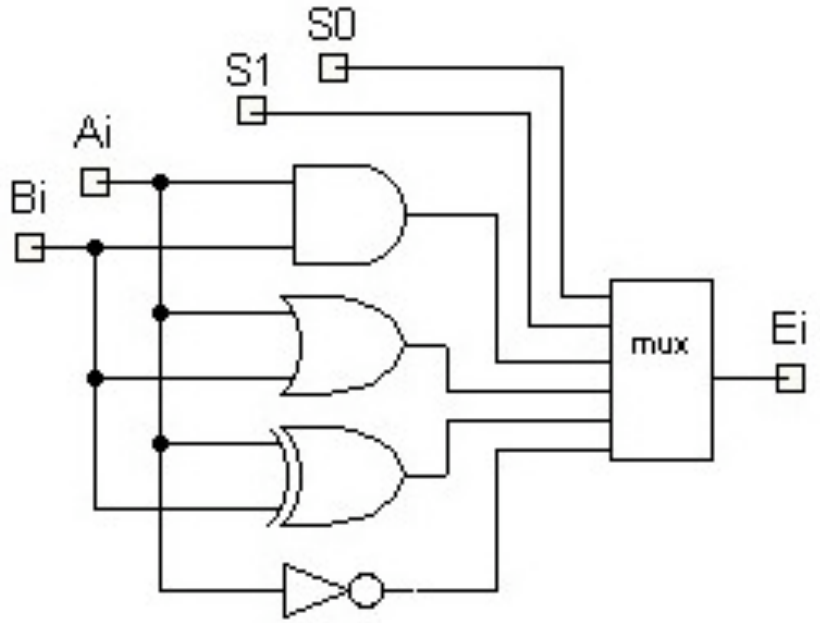
S_1 S_0 = 10 means shift left (divide: serial input for A_3, next flip-flop for others)

S_1 S_0 = 11 means parallel load



Logic Micro Operations

- All 16 logic micro-operations can be implemented using 4 basic operations: AND, OR, XOR and complement



S1	S0	Output	Operation
0	0	$E_i = A_i \wedge B_i$	AND
0	1	$E_i = A_i \vee B_i$	OR
1	0	$E_i = A_i \oplus B_i$	XOR
1	1	$E_i = \overline{A_i}$	Complement

Logic Micro Operations

- Logic micro-operations are carried out bitwise on registers, therefore this whole circuit is repeated for every bit of the register.
  - i.e. if register A has 8 bits, then there are 8 copies, one for each A_i , i = 0 ... 7
- The selection variables are applied to all stages simultaneously

- E.g. A = 01101001
  B = 10101010

Then

A ∧ B = 00101000

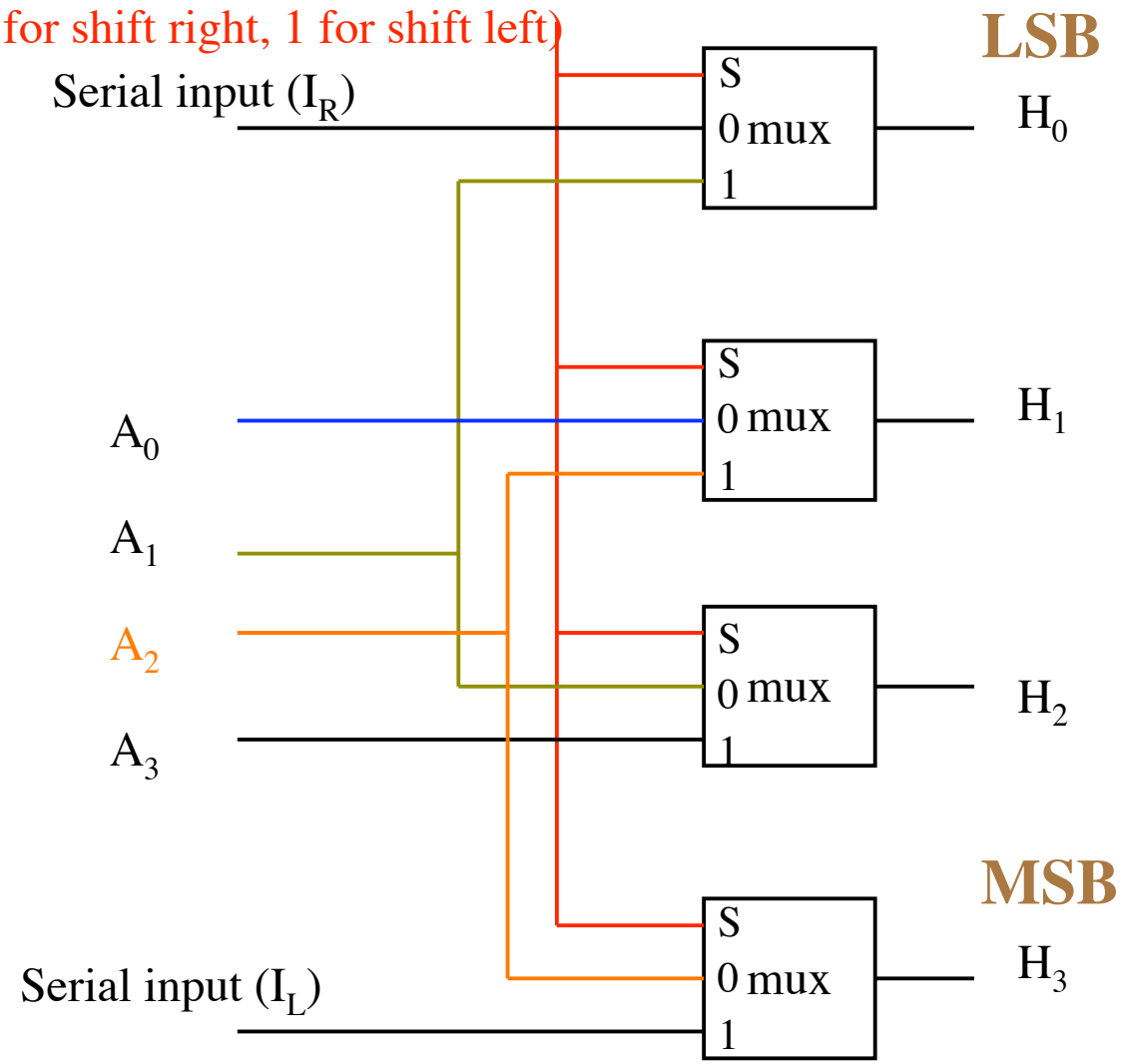
Shift Operations

- Bidirectional shift register (of slide 122) - but this needs 1 clock pulse to load the data and another to make the shift
  - too slow
- A combinational circuit is more efficient
- The register to be shifted is placed on a common bus connected to the shifter, and the shifted number is loaded into the register, taking only 1 clock pulse

Shift Operations(2)

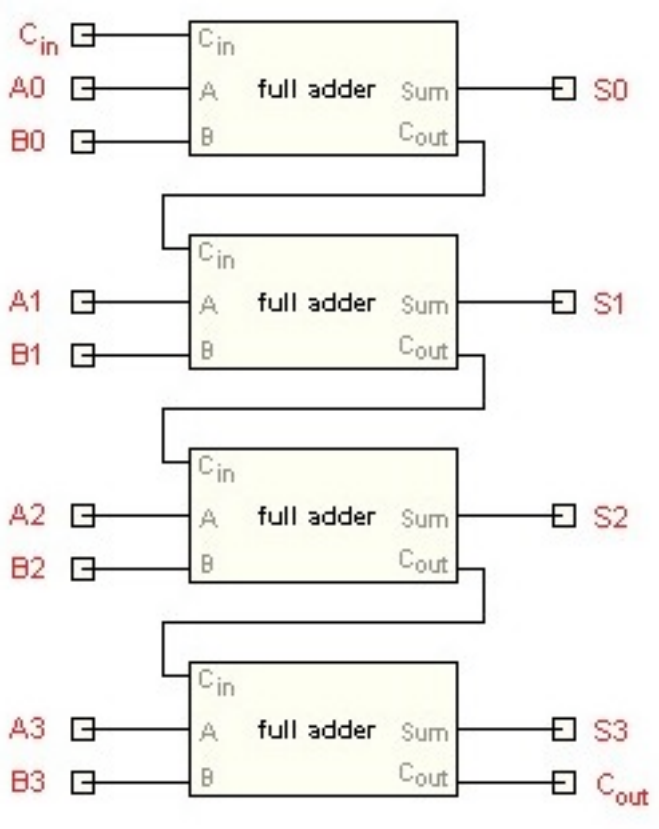
Function table

select	output			
s	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L



Arithmetic ops: Binary adder

- Calculates the arithmetic sum of 2 binary numbers of any length
- Constructed using a cascade of full adders (which add 2 bits and a carry)
- subscript 0 is least significant bit
- *n* bit binary adder requires *n* full adders (or *n-1* full adders and one half adder)



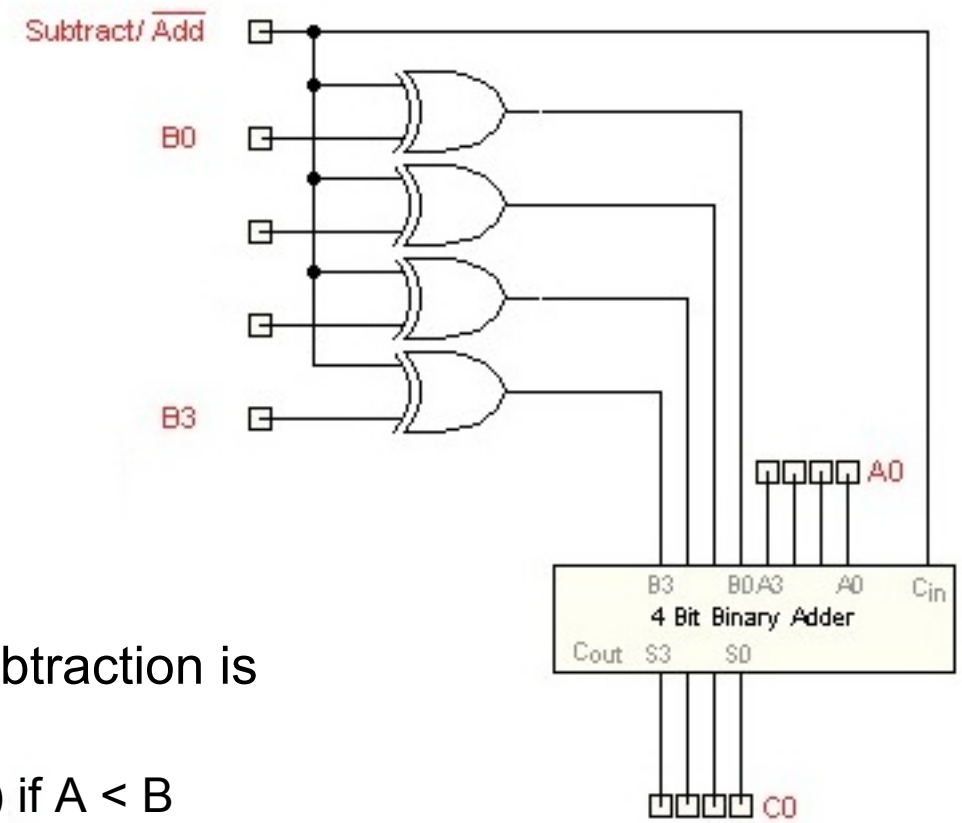
Binary Adder-Subtractor

- implements + and - together

Subtract/Add controls the mode (+ or -)

If 0, the circuit is an adder, ($B_i \oplus 0 = B_i$)

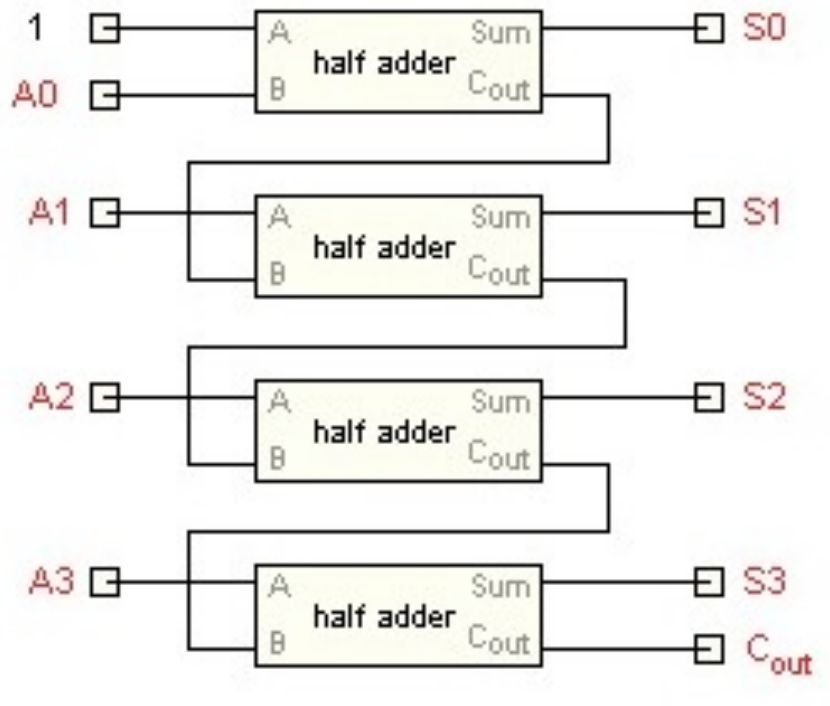
If 1, the circuit is a subtractor ($B_i \oplus 1 = B_i'$)
(and carry in $C_0 = 1$)



- For unsigned numbers, subtraction is
  - A - B if A ≥ B,
  - 2's complement of (B- A) if A < B
- For signed numbers, subtraction is A - B if there is no overflow

Binary Incrementer

- implements + 1
- Option 1 - use a counter
  - when the clock is enabled the contents of the register goes up by 1
- Option 2 - independent of register
  - Use *n* half adders in cascade



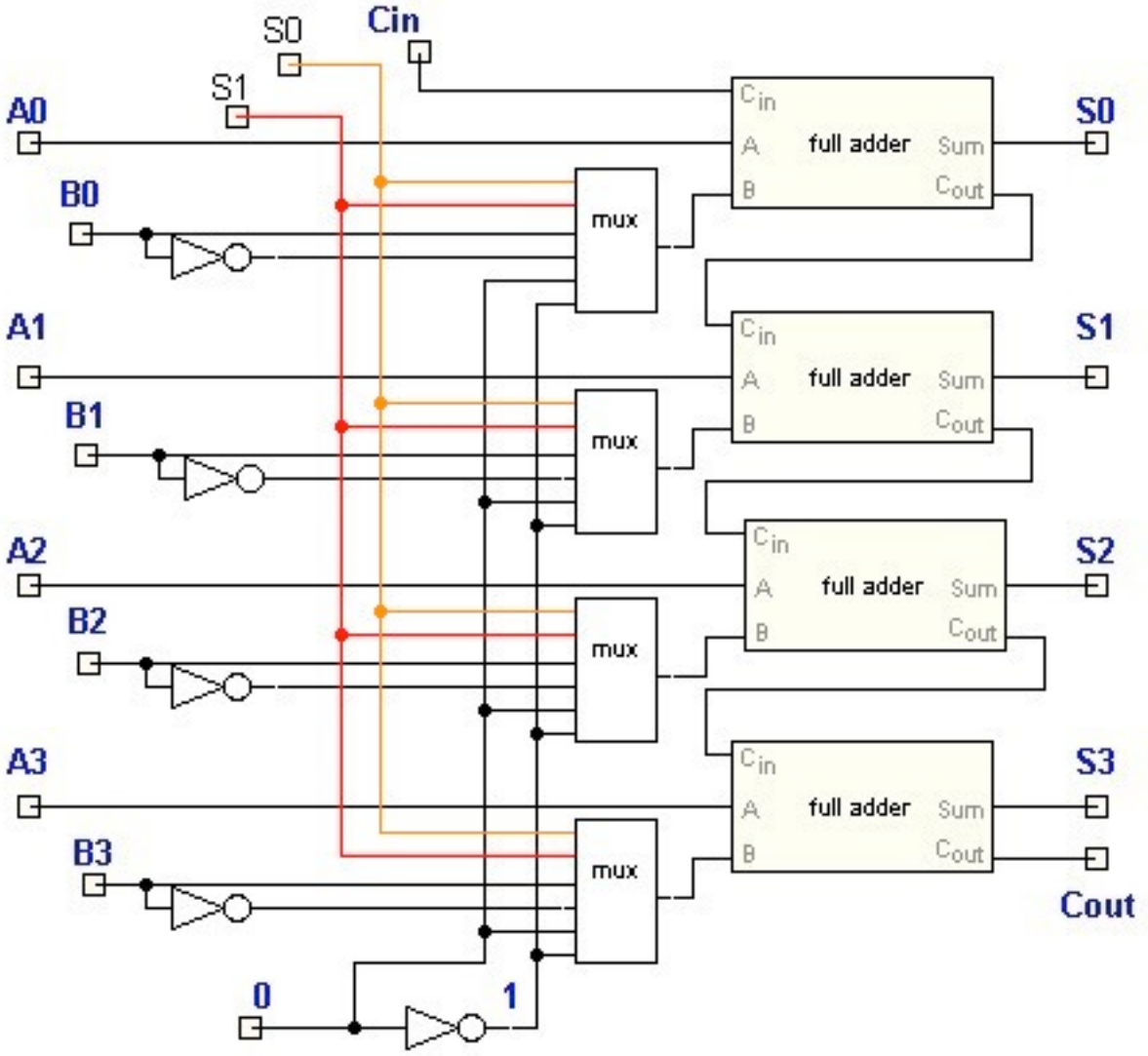
Arithmetic Circuit

- All operations can be implemented by a composite circuit (in the same way as all logic micro-operations were implemented in a single circuit)
- The circuit calculates

$$D = A + Y + C_{in}$$

- The multiplexers select the function
  - by varying Y
  - and the carry

Arithmetic Circuit



Arithmetic Circuit Functions

Function Table

Select			input	output	
S1	S0	Cin	Y	D=A + Y + Cin	micro-operation
0	0	0	B	D = A + B	add
0	0	1	B	D = A + B + 1	add with carry
0	1	0	$\overline{B}$	D = A + $\overline{B}$	subtract with borrow (A - B - 1)
0	1	1	$\overline{B}$	D = A + $\overline{B}$ + 1	subtract
1	0	0	0	D = A	transfer A
1	0	1	0	D = A + 1	increment A
1	1	0	1	D = A - 1	decrement A
1	1	1	1	D = A	transfer A

ignore B and just give 1s

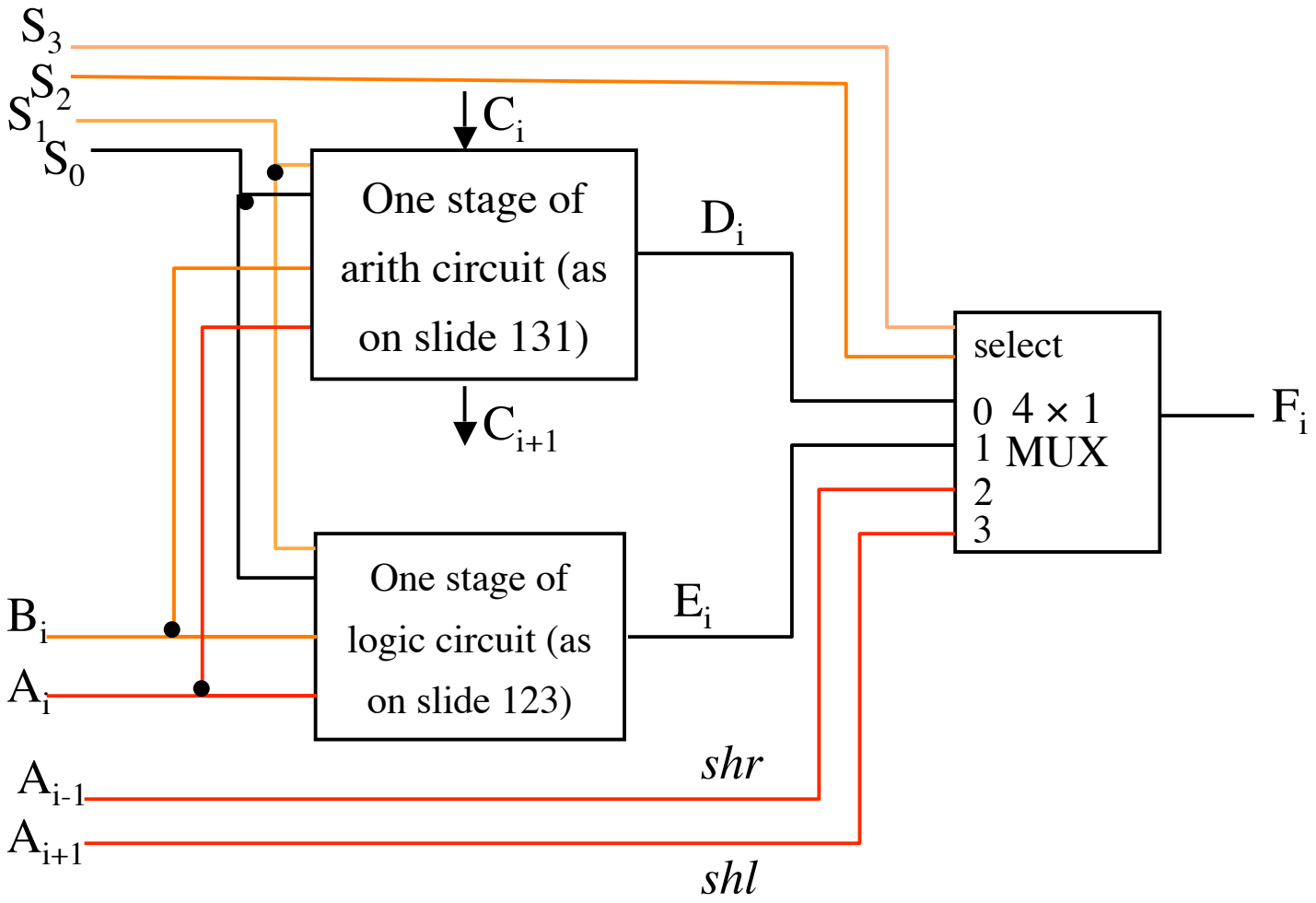
ignore B and just give 0s

For example, when subtracting A(2n) from B(2n) use the circuit twice (or 2 instances hooked up)

Hardware: Arithmetic Logic Shift Unit

- Instead of having separate sub-circuits for each kind of micro-operation most computers have a specialised unit with associated storage registers
  - this is the **Arithmetic Logic Unit** (ALU)
  - often just the arithmetic & logic operations, but sometimes also the shift operation
- The next slide shows the circuitry for 1 bit - this is repeated for *n* bits of the register size
- The micro-operation is selected using S1 S0
- the output of the arithmetic unit or logic unit or shifts is selected by S3 S2
- A C_{in} also affects which operation is chosen

Hardware: Arithmetic Logic Shift Unit



Function Table

S3	S2	S1	S0	Cin	Operation	Function
0	0	0	0	0	F = A + B	addition
0	0	0	0	1	F = A + $\overline{B}$ + 1	add with carry
0	0	0	1	0	F = A + $\overline{B}$	subtract with borrow
0	0	0	1	1	F = A + B + 1	subtraction
0	0	1	0	0	F = A	transfer A
0	0	1	0	1	F = A + 1	increment A
0	0	1	1	0	F = A - 1	decrement A
0	0	1	1	1	F = A	transfer A
0	1	0	0	x	F = A $\wedge$ B	AND
0	1	0	1	x	F = A $\vee$ B	OR
0	1	1	0	x	F = A $\oplus$ B	XOR
0	1	1	1	x	F = $\overline{A}$	complement A
1	0	x	x	x	F = shr A	shift right A into F
1	1	x	x	x	F = shl A	shift left A into F