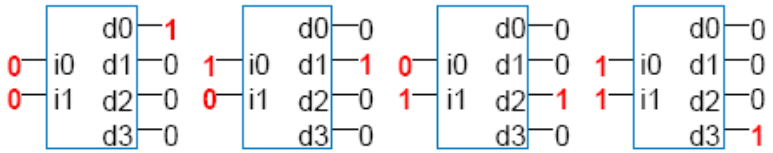
Digital Logic Design

Sung-Soo Lim

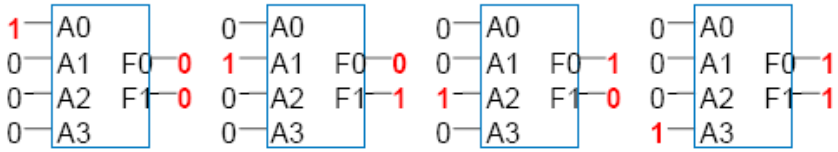
Code Converter

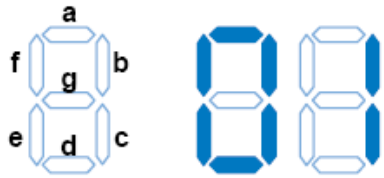
- Decoders
  - 이진 코드를 하나의 출력으로 변환하는 회로
- Encoders
  - 하나의 입력 비트를 이진 코드로 변환하는 회로
- Code Converters
  - 여러 코드 사이에 변환을 수행하는 회로
  - Example: BCD-to-7 Segment Display

2-to-4 Decoder

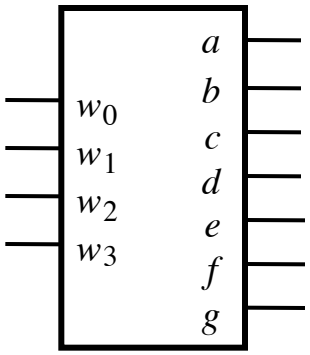


4-to-2 Encoder

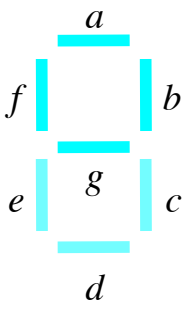




BCD-to-7 Segment Converter



(a) Code converter



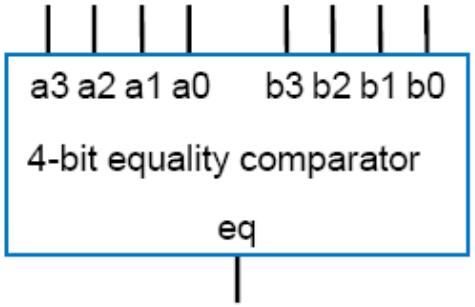
(b) 7-segment display

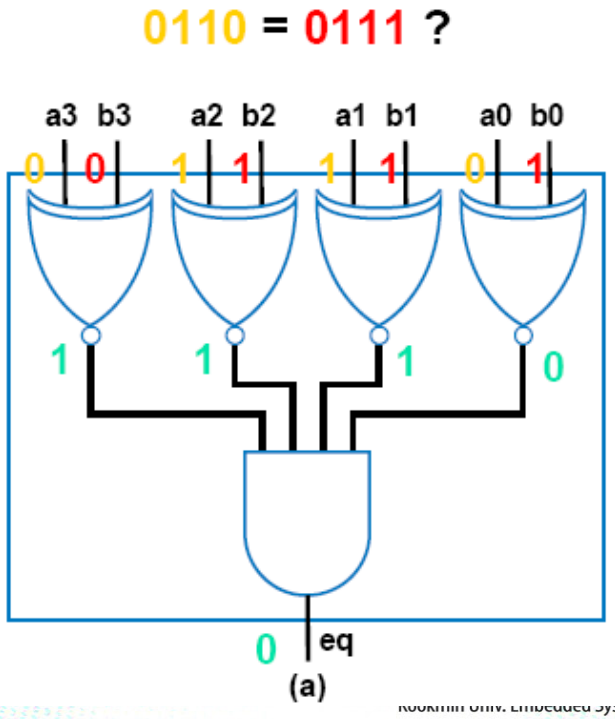
w_3	w_2	w_1	w_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

(c) Truth table

Equality Comparator

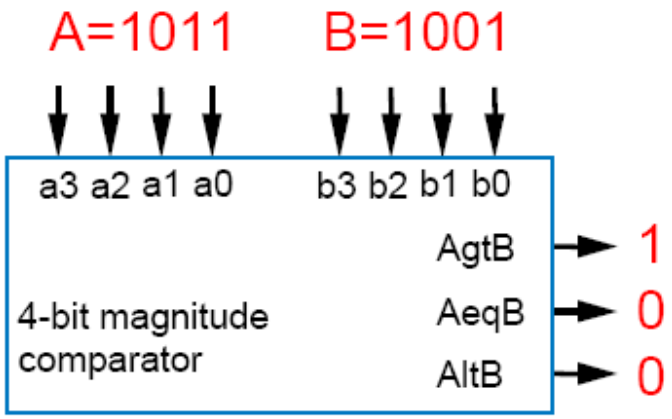
- N-bit equality comparator
  - 모든 N비트가 다 같은지 검사하는 회로
- 4-bit equality comparator with inputs A and B
  - For equality to be true
    - a3=b3, a2 = b2, a1 = b1, a0 = b0
    - Two bits are equal if both 1, or both 0
    - eq = (a3b3 + a3'b3') · (a2b2 + a2'b2') · (a1b1 + a1'b1') · (a0b0 + a0'b0')
  - XNOR 출력의 비밀
    - eq = (a3 xnor b3) · (a2 xnor b2) · (a1 xnor b1) · (a0 xnor b0)

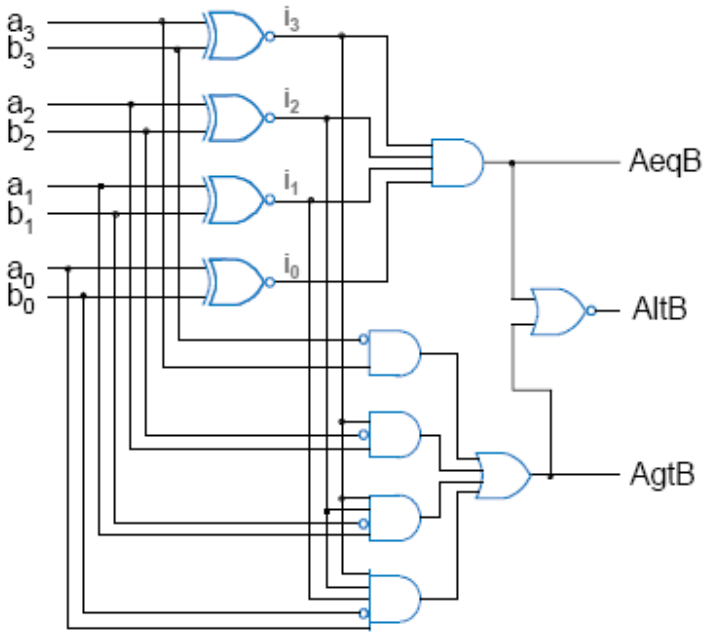




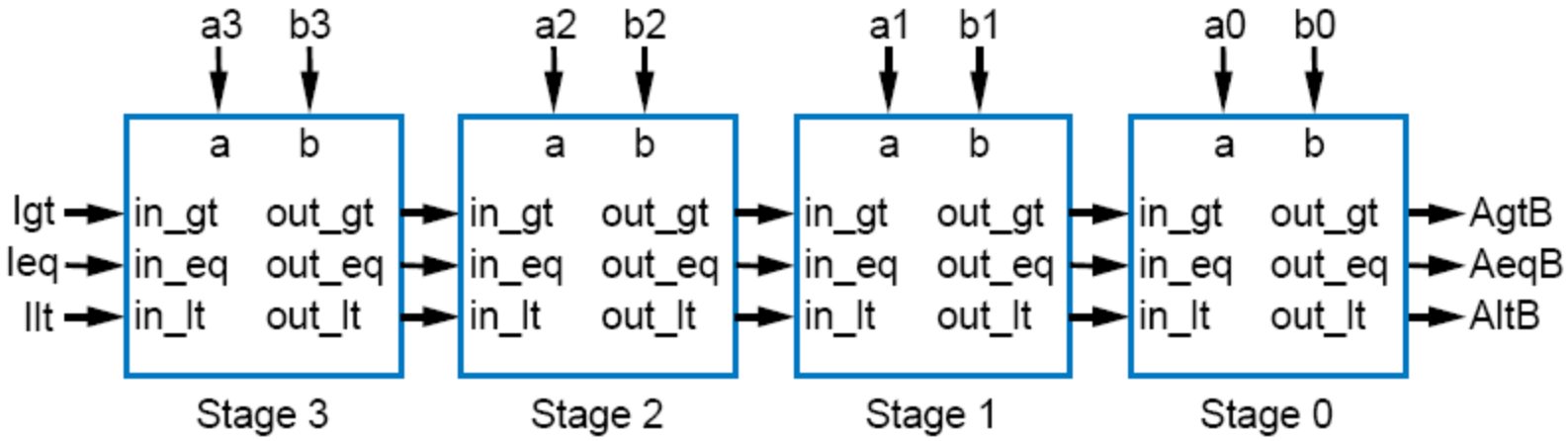
Magnitude Comparator

- N-bit magnitude (arithmetic) comparator
  - 2개의 N-bit 입력 A and B
  - A와 B 크기 비교 결과 출력: A>B, A=B, or A<B
- Implementation?
  - XNOR 게이트의 역할을 활용
    - $i_n = a_n$ xnor b_n
  - A = B, if all bits equal
    - AeqB = $i_3 i_2 i_1 i_0$
  - A > B
    - AgtB = $a_3 b_3' + i_3 a_2 b_2' + i_3 i_2 a_1 b_1' + i_3 i_2 i_1 a_0 b_0'$
      - a3 = 1, b3 = 0 ($a_3 b_3'$)
      - a3 = b3 and a2=1, a2=1 ($i_3 a_2 b_2'$)
      - and so on …
  - A < B
    - If A≠B and (A>B)' then must be A<B
    - AltB = (AeqB + AgtB)'





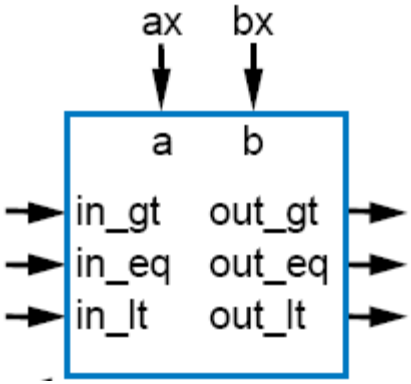
Magnitude Comparator (Alternative)

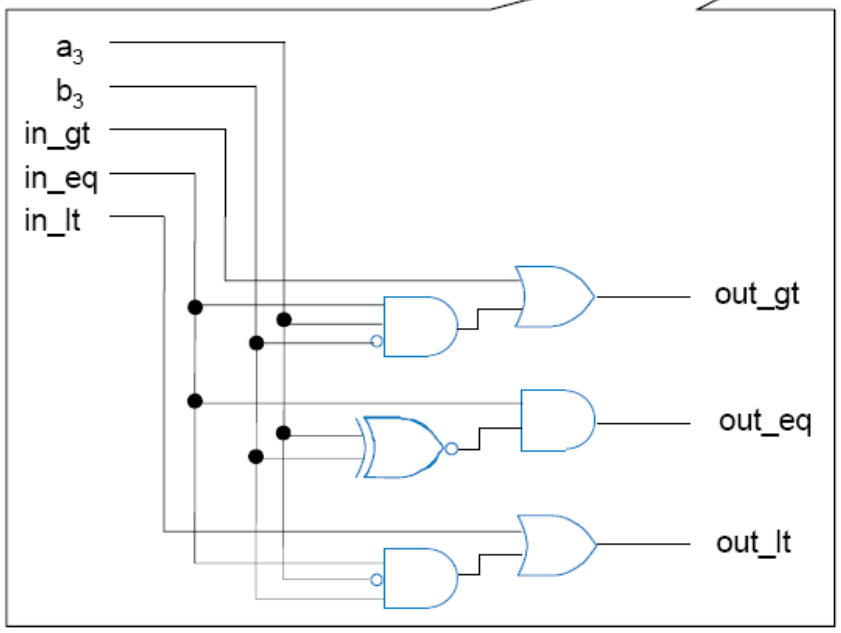


- Each stage:
  - out_gt = in_gt + (in_eq · a · b')
    - A>B (so far)
  - out_lt = in_lt + (in_eq · a' · b)
    - A<B (so far)
  - out_eq = in_eq · (a XNOR b)
    - A=B (so far)

Magnitude Comparator Internal Design

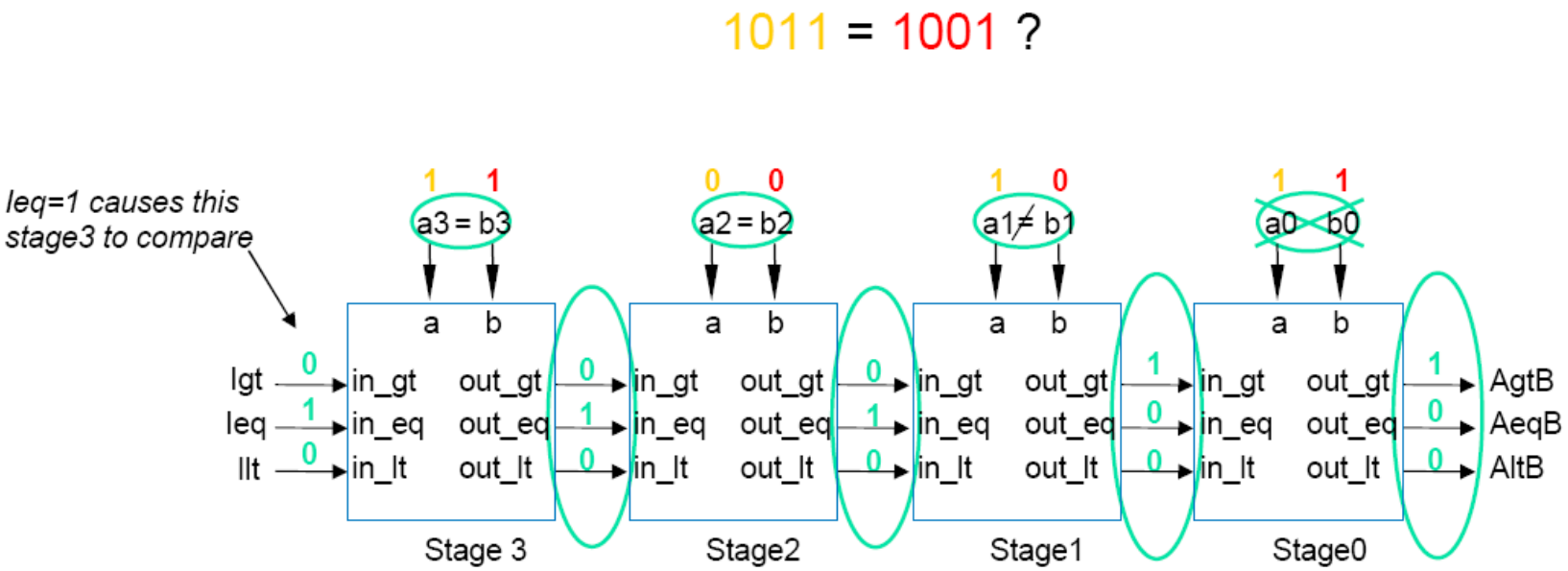
- 각 부분 회로의 내부 구조
  - out_gt = in_gt + (in_eq · a · b')
  - out_eq = in_eq · (a XNOR b)
  - out_lt = in_lt + (in_eq · a' · b)





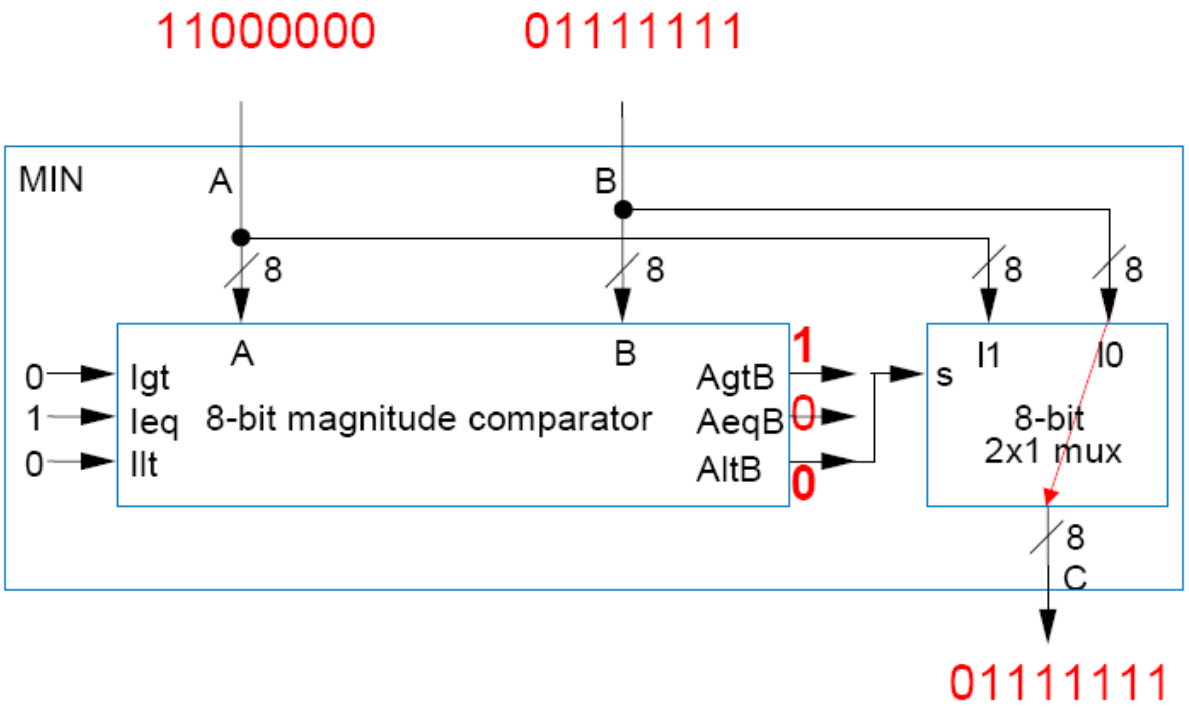
Magnitude Comparator In Action

- How does it work?
  - 마지막 결과는 맨 마지막 회로를 통해 출력
  - 어디선가 본 듯한 스타일?



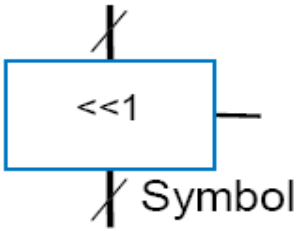
Minimum Detection

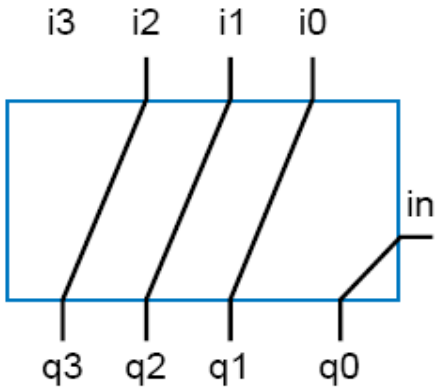
- 두 개의 8비트 입력 중 작은 수를 골라내는 회로
  - Solution: 8-bit magnitude comparator 와 8-bit 2x1 mux 사용
  - If A<B, A 를 Mux로. 아닌 경우, B를 Mux로.



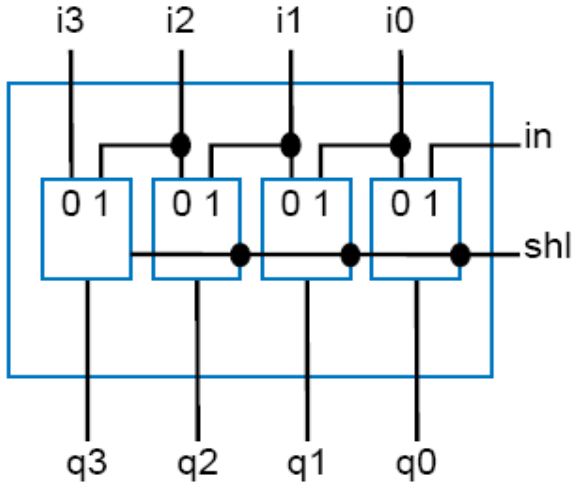
Left Shifter

- Left Shift
  - 1비트 왼쪽 시프트: multiplying by 2
    - 0011 (3) becomes 0110 (6)
    -





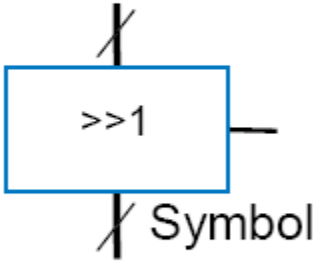
Left shifter (just wires)

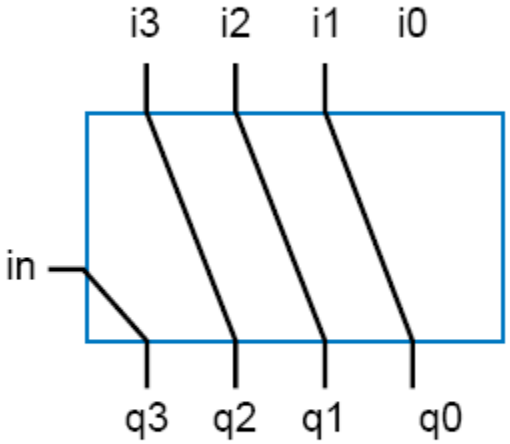


Shifter with left shift or no shift using MUXes

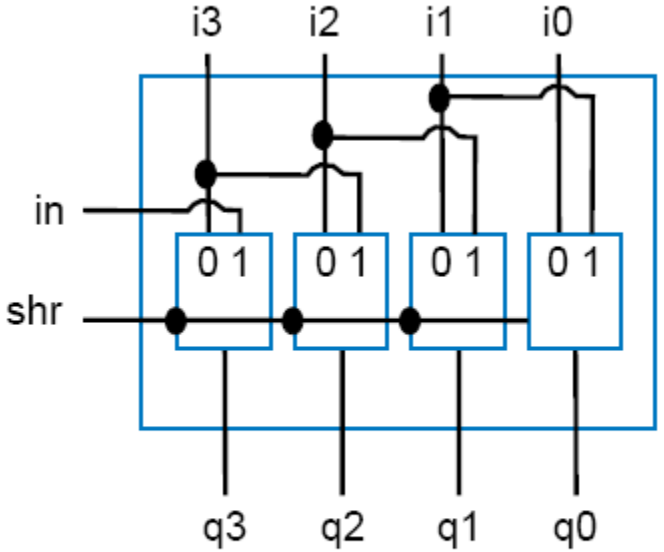
Right Shifter

- Shift Right
  - 1비트 오른쪽 시프트: dividing by 2
  - 1000 (8) becomes 0100 (4)





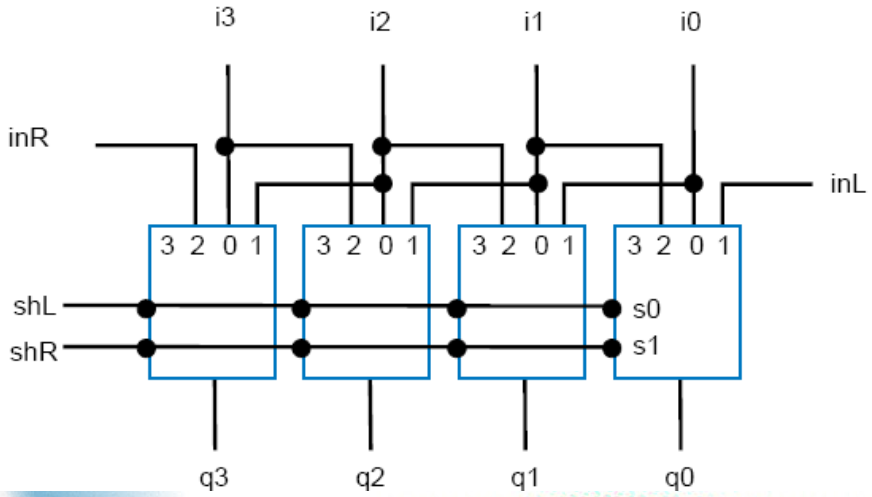
Right shifter (just wires)



Shifter with right shift or no shift using MUXes

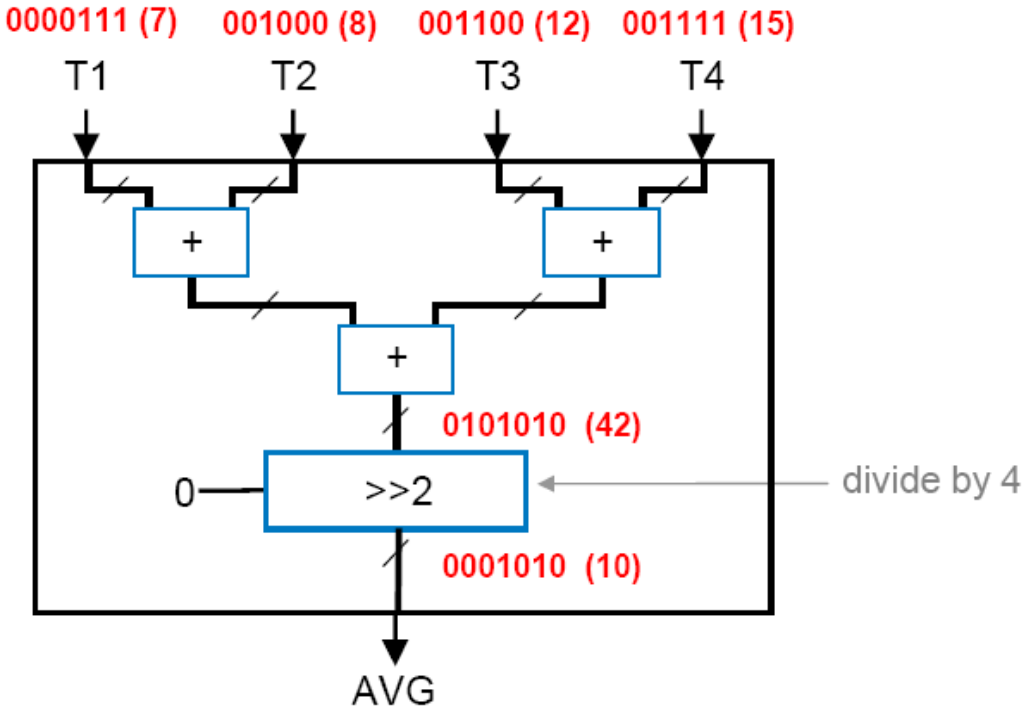
Multifunction Shifter

- Multifunction Shifter - 왼쪽, 오른쪽, 혹은 시프트 없이 그대로 출력
  - MUX 사용
  - Operation
    - shL = 0 and shR = 0, no shift
    - shL = 0 and shR = 1, shift left
    - shL = 1 and shR = 0, shift right
    - shL = 1 and shR =1, unused



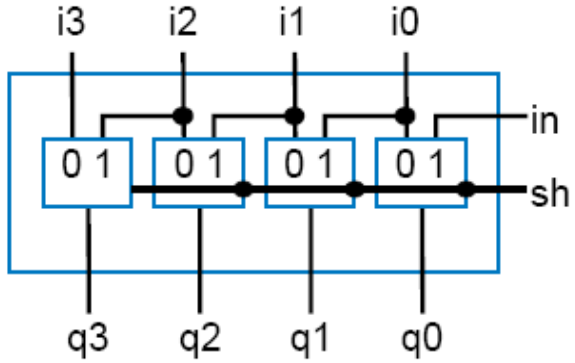
Shifter Example: Temperature Averager

- Design a circuit to compute the average four temperatures
  - Add, then divide by four
  - Same as shift right by 2
  - Use three adders, and right shift by two

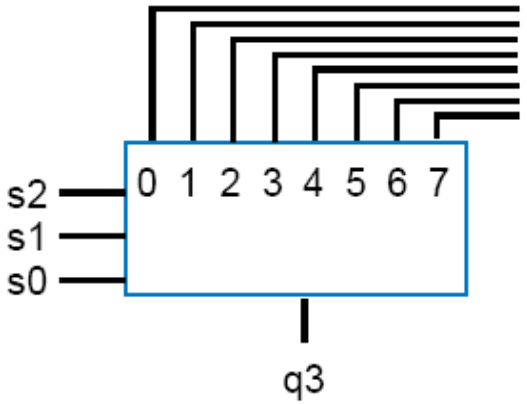


Barrel Shifter

- 주어진 비트수만큼 시프트 연산을 수행하는 회로
  - 4-bit barrel left shift : shift left by 0, 1, 2, or 3 positions
  - 8-bit barrel left shifter: shift left by 0,1, 2, 3, 4, 5, 6, or 7 positions
- 8x1 MUXes 와 여러 선을 이용해 구현 가능
  - 선이 너무 많다



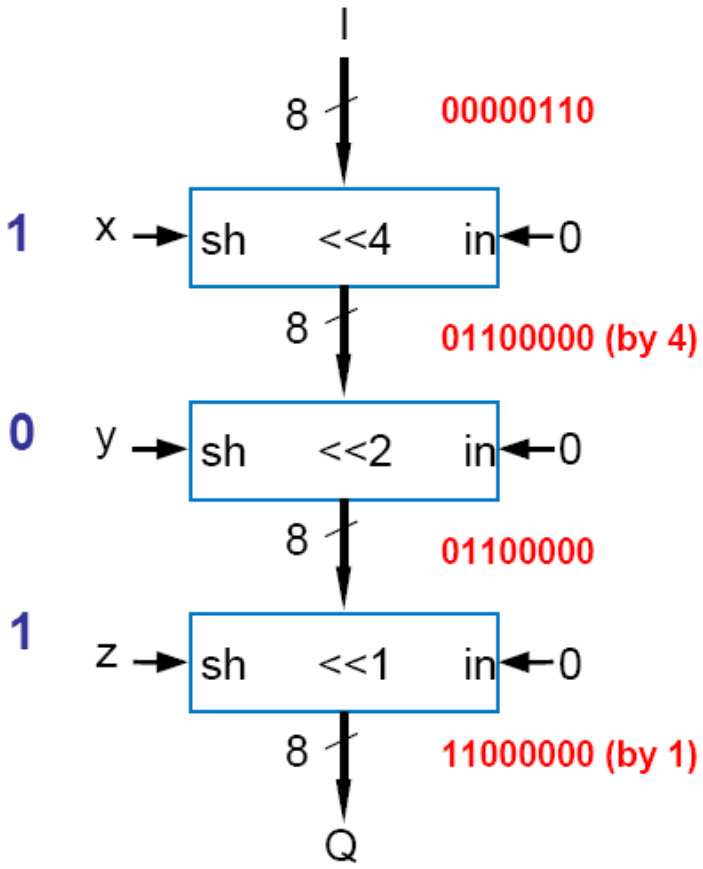
Shift by 1 shifter uses 2x1 muxes.



8x1 mux solution for 8-bit barrel shifter: too many wires.

Barrel Shifter - Revised

Q: xyz to shift by 5?



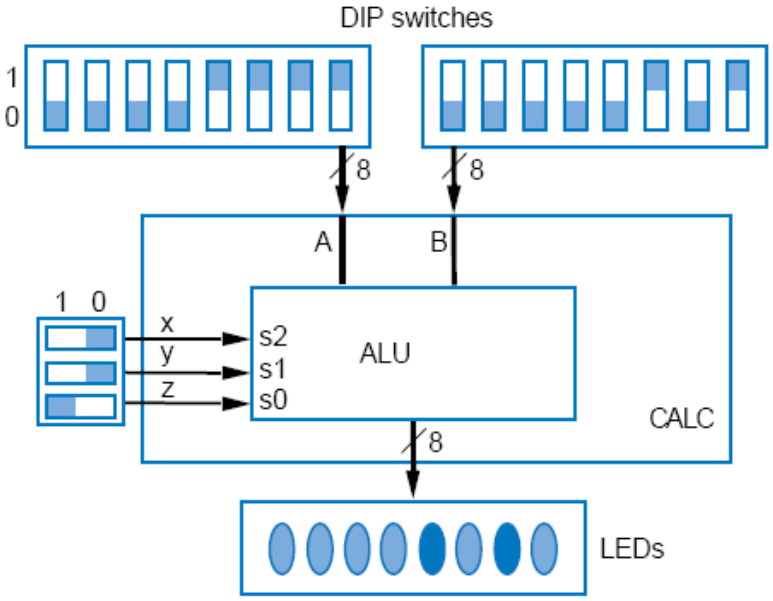
Net result: shift by 5

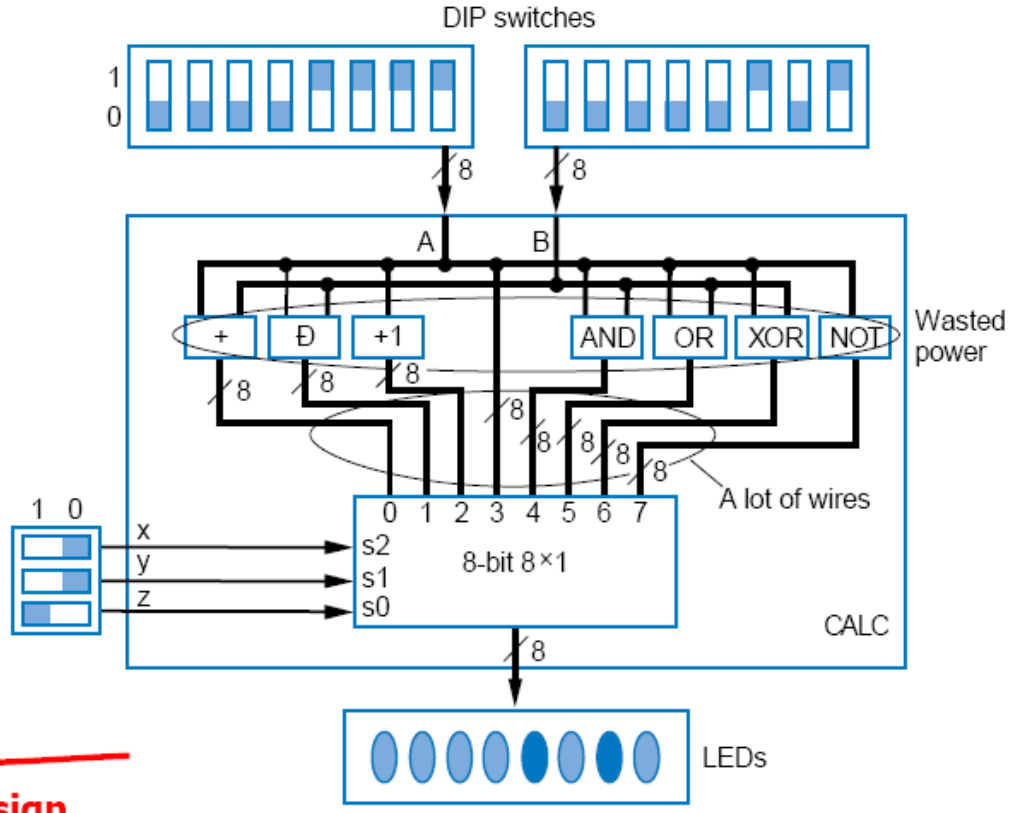
ALU

- Arithmetic Logic Unit (ALU)
  - 여러 산술 연산 (add, subtract, increment, etc.) 및 논리 연산 (AND, OR, etc.) 수행: 연산을 선택하는 명령어 비트 존재
- Motivation

Inputs			Outputs	Sample output if A=0000 1111, B=0000 0101
x	y	z		
0	0	0	S = A + B	S = 0001 0100
0	0	1	S = A - B	S = 0000 1010
0	1	0	S = A + 1	S = 0001 0000
0	1	1	S = A	S = 0000 1111
1	0	0	S = A AND B (bitwise AND)	S = 0000 0101
1	0	1	S = A OR B (bitwise OR)	S = 0000 1111
1	1	0	S = A XOR B (bitwise XOR)	S = 0000 1010
1	1	1	S = NOT A (bitwise complement)	S = 1111 0000

Multi-Function Calculator





ALU simplifies design, no wasted power