

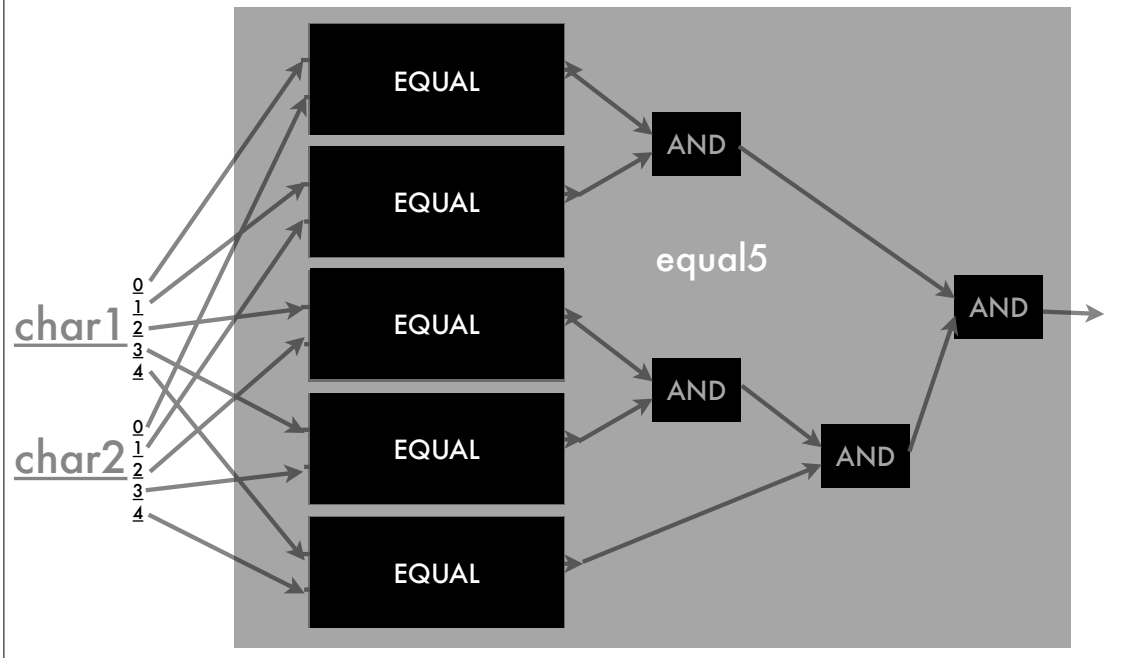
Lecture 4: Binary

CS442: Great Insights in Computer Science
Michael L. Littman, Spring 2006

I-Before-E, Continued

- There are two ideas from last time that I'd like to flesh out a bit more.
- This time, let's proceed bottom up.

Equal5 Diagram



Gates in EQUAL5

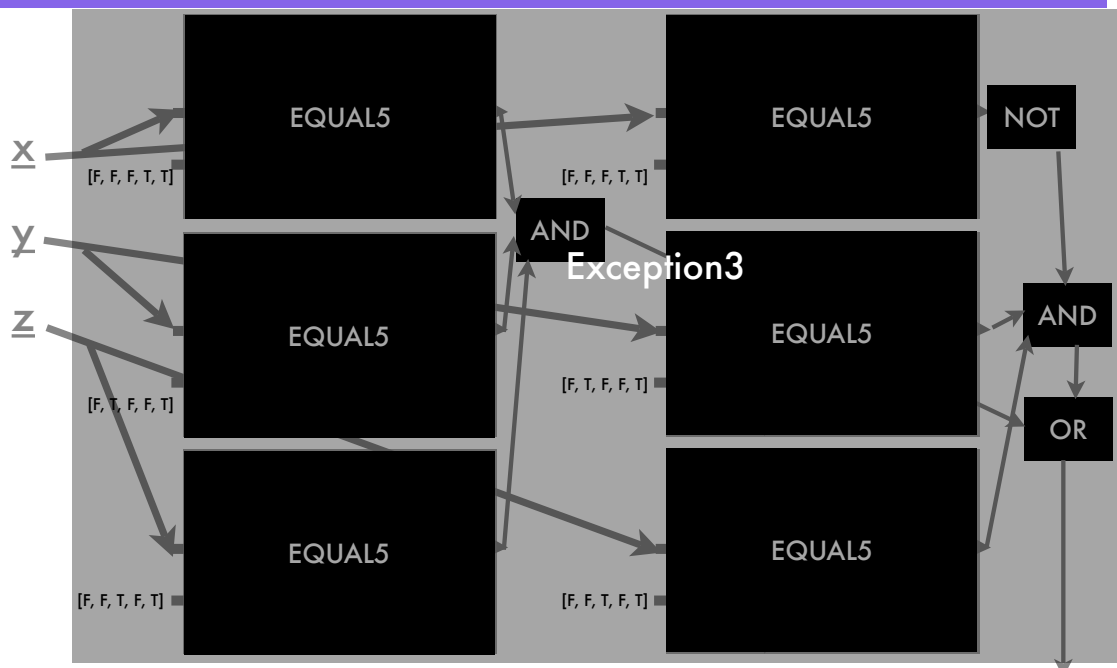
- 10 inputs (2 groups of 5), 1 output bit.
- The equal5 gate consists of
 - 4 “and” gates
 - 5 “equal” gates
 - 2 “and”, 2 “not”, 1 “or” (5 total)
 - Total = 29 gates

Local Exception Check

```
# [False, False, False, True, True] is 'c'  
# [False, True, False, False, True] is 'i'  
# [False, False, True, False, True] is 'e'  
def exception3(x,y,z):  
    ex1 = AND3(equal5(x,[False, False, False, True, True]),  
               equal5(y,[False, True, False, False, True]),  
               equal5(z,[False, False, True, False, True]))  
    ex2 = AND3(not equal5(x, [False, False, False, True, True]),  
               equal5(y,[False, False, True, False, True]),  
               equal5(z, [False, True, False, False, True]))  
    return ex1 or ex2
```

- A triple of letters is an exception if it is equal to "cie" or "*ei" where * is not equal to c.

Exception3 Diagram



Gates in Exception5

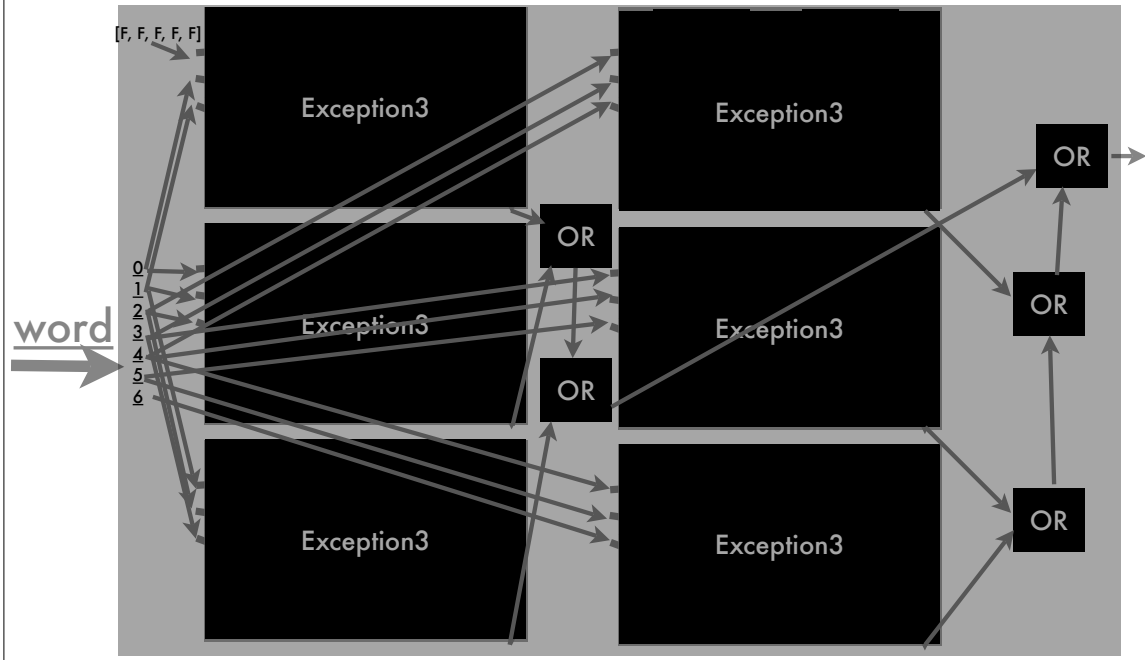
- 15 inputs (3 groups of 5), 1 output bit.
- The Exception5 gate consists of
 - 2 “and” gates, 1 “not”, 1 “or” (4 total)
 - 6 “equal5” gates
 - 29 total basic gates per gate
 - Total = 178 gates.

Finally, The Whole Thing

```
# word is 7 groups of 5 bits each
# [False, False, False, False, False] is '_'
def exception(word):
    ps = exception3([False, False, False, False, False], word[0], word[1])
    p0 = exception3(word[0], word[1], word[2])
    p1 = exception3(word[1], word[2], word[3])
    p2 = exception3(word[2], word[3], word[4])
    p3 = exception3(word[3], word[4], word[5])
    p4 = exception3(word[4], word[5], word[6])
    return ((ps or p0) or p1) or (p2 or (p3 or p4))
```

- A seven-letter word is an exception if any of its 3-letter windows shows an exception.

Exception Diagram



Gates in Exception

- 35 inputs (7 groups of 5), 1 output bit.
- The Exception gate consists of
 - 5 “or” gates
 - 6 “exception3” gates
 - 178 total basic gates per gate
- Total = 1073 gates, without breaking a sweat.

Exception

- Let's try it out.
- <http://www.cs.rutgers.edu/~mlittman/courses/cs442-06/python/exception.py>

Counting (Decimal)

- How do we count?
- Start at the bottom digit.
 - If it's less than 9, add one to it.
 - If it's equal to nine, make it zero and proceed to the digit to the left.

2 4 9 9 8

Counting (Binary)

- Counting in binary is the same idea.
- Start at the bottom (rightmost) bit.
 - If it's less than 1, add one to it.
 - If it's equal to 1, make it zero and proceed to the bit to the left.

0 0 0 0 1

Place Values

- Because of the way counting works, we expand the representation by another bit for each power of 2.

- So, 01100101 is:

- $64+32+8+1=105$

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	1	1	0	1	0	0	1

Number Magic

- How does this trick work?
- <http://www.brainbashers.com/games/number.asp>

Which Have Your Number?

- Think of a number from 0 to 31.
- Add the upper left number from each card your number appears on.
- It is...

16	17	18	19	8	9	10	11	4	5	6	7	2	3	6	7	1	3	5	7
20	21	22	23	12	13	14	15	12	13	14	15	10	11	14	15	9	11	13	15
24	25	26	27	24	25	26	27	20	21	22	23	18	19	22	23	17	19	21	23
28	29	30	31	28	29	30	31	28	29	30	31	26	27	30	31	25	27	29	31

Conversion

- To go from decimal to binary, start with the biggest power of 2 no bigger than your number.
- Write down a 1. Subtract the power of 2 from your number.
- Cut the power of 2 in half.
- If your remaining number is larger than the power of 2, write down a 1 and subtract the power of 2.
- If not, write down 0.
- Repeat by cutting the power of 2 in half (until you get to 1).

Example: Convert 651

- Bigger than: $2^9 = 512$. **1**
 - $651 - 512 = 139$.
 - Next power of 2 = 256. **0**
 - Next power of 2 = 128. **1**
 - $139 - 128 = 11$.
 - Next power of 2 = 64. **0**
 - Next power of 2 = 32. **0**
 - Next power of 2 = 16. **0**
 - Next power of 2 = 8. **1**
 - $11 - 8 = 3$
 - Next power of 2 = 4. **0**
 - Next power of 2 = 2. **1**
 - $3 - 2 = 1$
 - Last power of 2 = 1. **1**
- 1010001011 = 651**

Binary Addition

$$\begin{array}{r} 01110011 \\ + 10110010 \\ \hline \end{array} \quad \begin{array}{r} 115 \\ +178 \\ \hline 293 \end{array}$$

- Just like in school: work left to right, carry when needed.
- $0+0+0=0$, $0+0+1 = 1$, $0+1+1=10$, $1+1+1=11$
- Can check via conversion.

Other Operations

- Can also define subtraction (with borrowing), multiplication (simpler since there are only 3 facts: $0 \times 0 = 0$ $0 \times 1 = 0$ $1 \times 1 = 1$, look familiar?), and long division.
- Can do bitwise logic operations (and, or, not).
- All are quite useful...

Other Number Schemes

- Can represent negative numbers, often via complements. $-1 = 256-1 = 255$.
- Fixed-width fractions (for dollar amounts).
- Floating point representations via exponential notation: $a \times 10^b$.
- Complex numbers: real and imaginary parts.

Implementing Addition

- Half adder: Takes two bits and a carry and outputs a bit and a carry (addc).
- Adder: Adds two 8-bit numbers (discards last carry) (addbyte).

```
def addc(a,b,c):
```

```
    bit = (a and not b and not c) or (not a and b and not c) or (not a and not b and c) or (a and b and c)
    carry = (a and b and not c) or (a and not b and c) or (not a and b and c) or (a and b and c)
    return([carry, bit])
```

```
def addbyte(x,y):
```

```
    z = [0]*8
    sum7 = addc(x[7],y[7],0)
    z[7] = sum7[1]
    sum6 = addc(x[6],y[6],sum7[0])
    z[6] = sum6[1]
    sum5 = addc(x[5],y[5],sum6[0])
    z[5] = sum5[1]
    sum4 = addc(x[4],y[4],sum5[0])
    z[4] = sum4[1]
    sum3 = addc(x[3],y[3],sum4[0])
    z[3] = sum3[1]
    sum2 = addc(x[2],y[2],sum3[0])
    z[2] = sum2[1]
    sum1 = addc(x[1],y[1],sum2[0])
    z[1] = sum1[1]
    sum0 = addc(x[0],y[0],sum1[0])
    z[0] = sum0[1]
    return z
```

Difference Engine

- Takes a list of 5 numbers: column from the earlier representation.
- Produce the next column.
- Output of circuit is input of next iteration!

```
def DE(numlist):  
    [a1,b1,c1,d1,e1] = numlist  
    e2 = e1  
    d2 = addbyte(d1,e1)  
    c2 = addbyte(c1,d1)  
    b2 = addbyte(b1,c1)  
    a2 = addbyte(a1,b1)  
    return [a2,b2,c2,d2,e2]
```



Next Time

- We now have all the pieces to build a simple, working computer...
 - Each cycle, inputs propagate to outputs, which are copied back to inputs to begin again.
- We need a language to talk to it in, though.
- Read Hillis Chapter 3.