Introduction to Programming for Scientists

Lecture 4

More programming examples, file manipulation, numbers

Prof. Steven Ludtke
N410, sludtke@bcm.edu

HOMEWORK REVIEW

```
loan=int(raw_input("Enter loan amount: "))
rate=(float(raw_input("Enter annual interest rate: "))/100)*(1.0/12.0)
payment=int(raw_input("Enter payment amount: "))

counter=1
results=file("results.txt","w")
while loan > 0:
        interest=loan*rate
        if (loan < payment):
                results.write("%3d: Interest:%8.2f Balance:%10.2f\n"%(counter, interest, 0.0))
                break
        loan=interest+loan-payment
        results.write("%3d: Interest:%8.2f Balance:%10.2f\n"%(counter, interest, loan))
        counter=counter+1

results.close()
```

‘Scrabble’ Problem

- What words could you make given these letters: PGAORRM ?

‘Scrabble’ Problem

- What words could you make given tiles containing PGAORRM
- 68 of them
- program armor gompa gramp groma maror morra pargo gamp gapo gora gorm gorp gram marg mora ogam orra parr pram prao proa prog prom ramp roam roar roma romp ago amp apo arm gam gap gar goa gor mag map mar moa mog mop mor oar ora pam par poa pom pro rag ram rap rom ag am ar go ma mo om op or pa po a

‘Scrabble’ Problem

- You have 7 random letters. What real words can you make from them ?
- How many ‘words’ could we make ?
- 7*6*5*4*3*2*1 + 7*6*5*4*3*2 + ... = 7! + 7!/1! + 7!/2! + 7!/3! + ... = 13699

‘Scrabble’ Problem

- You have 7 random letters. What real words can you make from them ?
- How many letter combinations could we make ?
- 7*6*5*4*3*2*1 + 7*6*5*4*3*2 + ... = 7! + 7!/1! + 7!/2! + 7!/3! + ... = 13699
- Different approaches:
  - Make all possible words, check to see if each is in the dictionary
    - Linear
    - Recursive
  - Check each word in the dictionary to see if it can be made from the letters in the list

Design #1

- ask for letters
- Read list of words
- Nested loop to make each possible word from letters
  - check to see if word is real
  - if so, add it to the list
- sort the list and print results

Design #2

- ask for letters
- Read list of words
- Recursion to make each possible word from letters
  - check to see if word is real
  - if so, add it to the list
- sort the list and print results

Recursion

- A function that calls itself

```
def factorial(x):
    if x==1 : return 1
    return x*factorial(x-1)
```

Design #3

- ask for letters
- Read list of words
- Loop over list of words
  - make a list of available letters
  - Loop over the letters in word
    - see if each letter is in the list, if so, remove
  - if we found all of the letters print the word

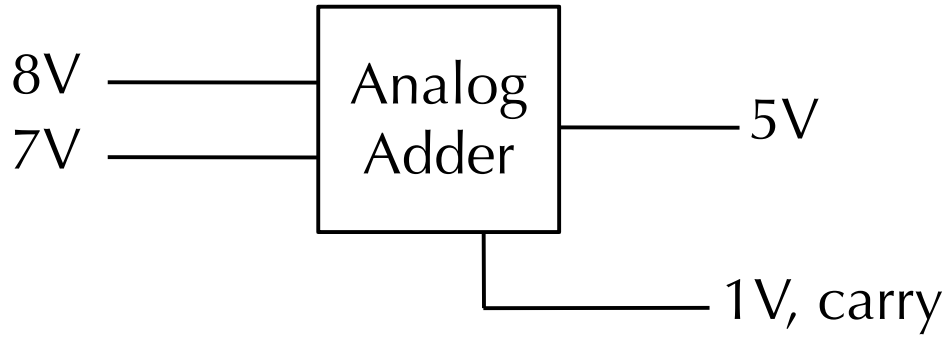
Results

- Nested loop : 93.8 sec
- Nested loop (set): 0.44 sec
- Recursive : 0.38 sec
- Check all words: 0.92 sec

Numbers & Computers

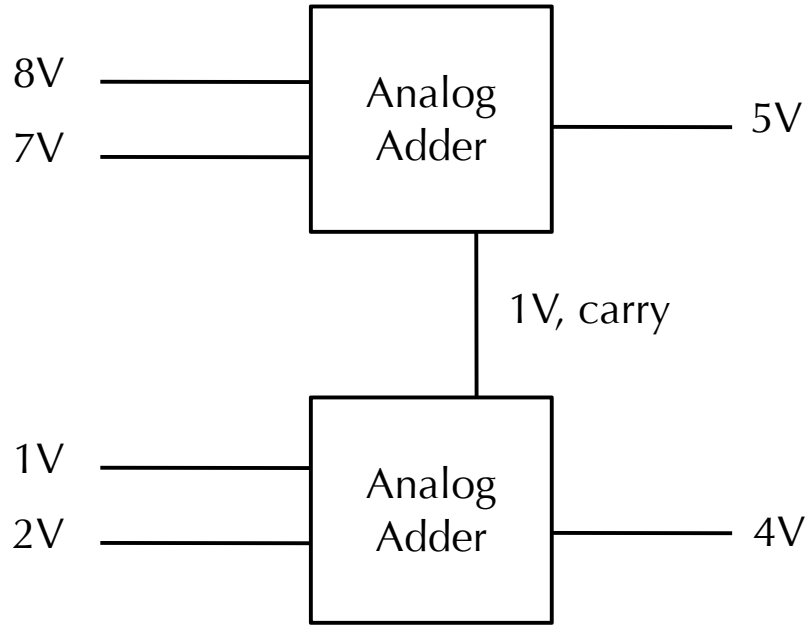
Why Binary ?

8+7=15

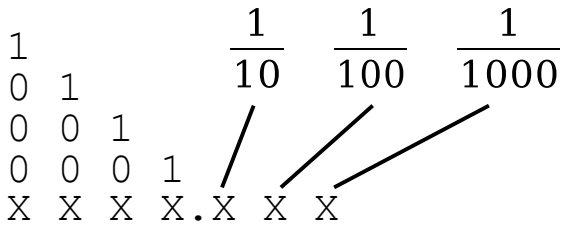


Why Binary ?

18+27=45



Decimal Numbers

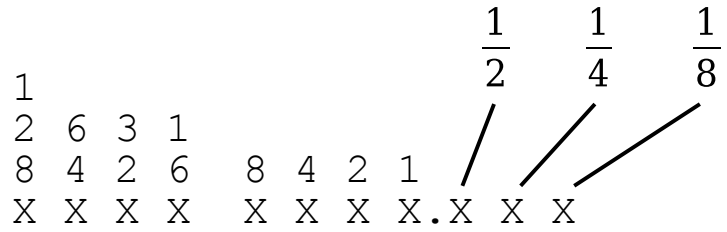


Binary Numbers

```
                1
                2  6  3  1
                8  4  2  6     8  4  2  1
                X  X  X  X     X  X  X  X
```

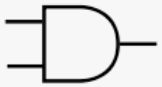
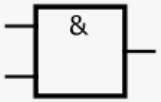
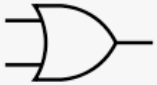
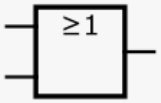
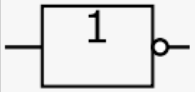
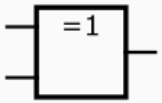
1	0000 0001
2	0000 0010
4	0000 0100
8	0000 1000
15	0000 1111
212	1101 0100

Binary Numbers



1	0000 0001
2	0000 0010
4	0000 0100
8	0000 1000
15	0000 1111
212	1101 0100
0.25	0000 0000.010
0.625	0000 0000.101
0.3	0000 0000.0100 1100 1100 ...

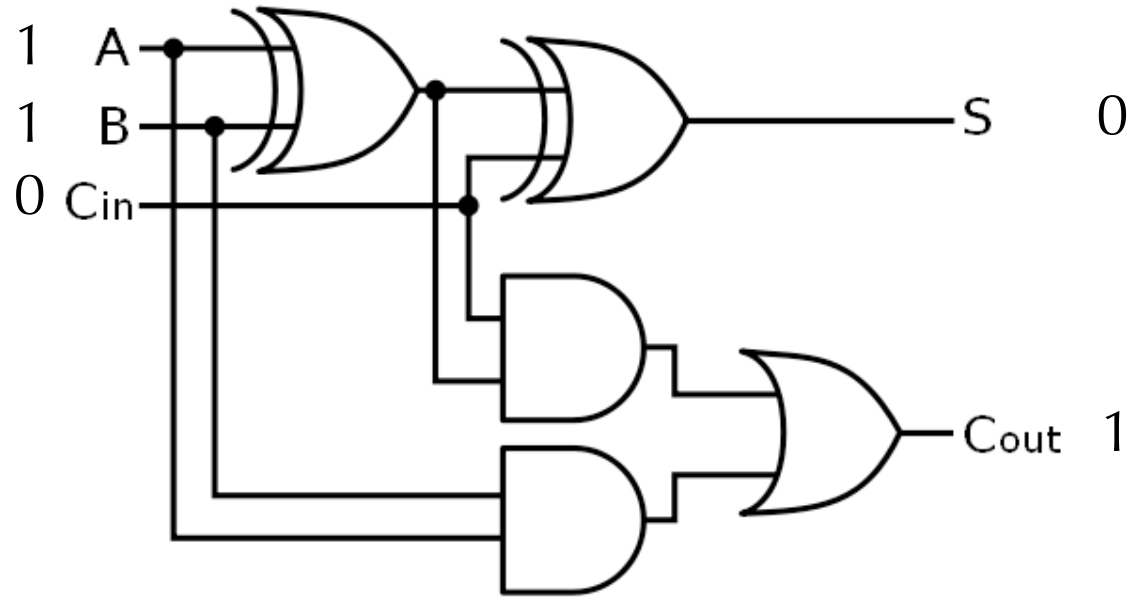
Boolean Algebra Made Real

	Distinctive shape	Rectangular shape	Boolean algebra between A & B	Truth table
and			$A \cdot B$	INPUT A B / OUTPUT A AND B: 0 0 → 0; 0 1 → 0; 1 0 → 0; 1 1 → 1
or			$A+B$	INPUT A B / OUTPUT A OR B: 0 0 → 0; 0 1 → 1; 1 0 → 1; 1 1 → 1
not			$\overline{A}$	INPUT A / OUTPUT NOT A: 0 → 1; 1 → 0
xor			$A \oplus B$	INPUT A B / OUTPUT A XOR B: 0 0 → 0; 0 1 → 1; 1 0 → 1; 1 1 → 0

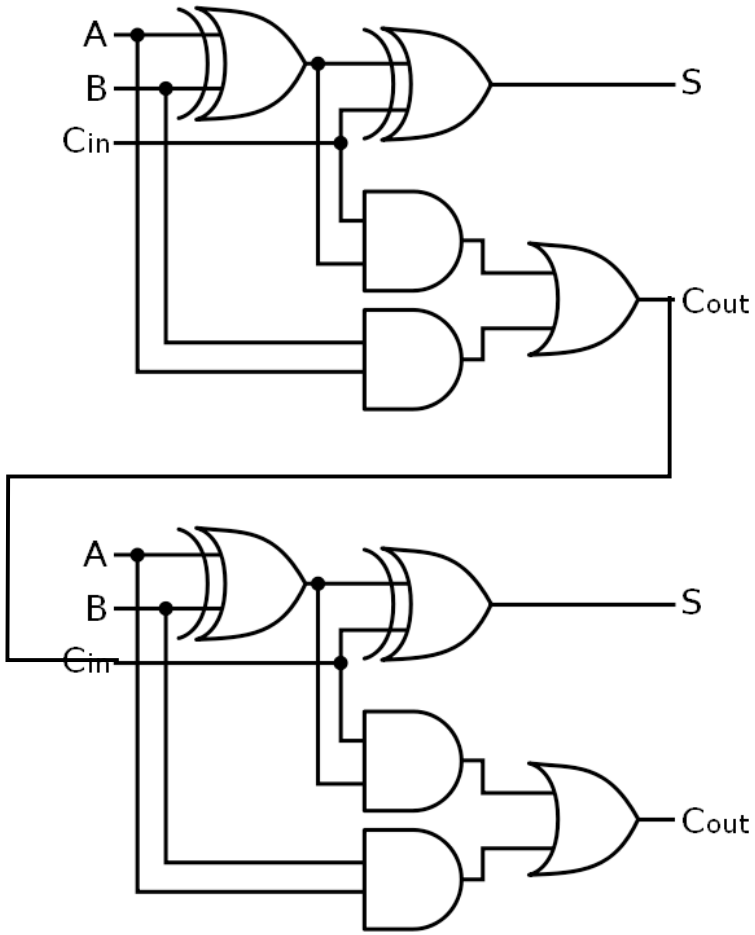
http://en.wikipedia.org/wiki/Logic_gates

1 bit binary adder

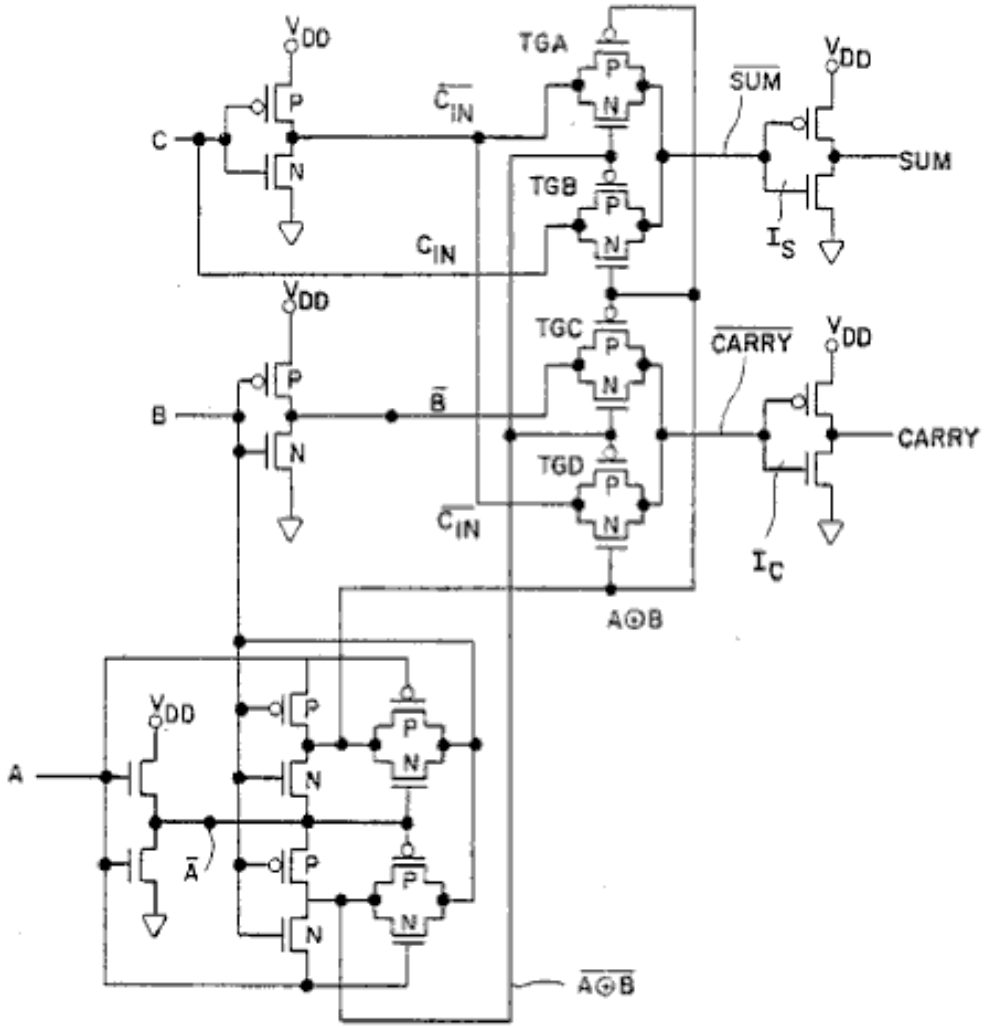
1+1=10



2 bit binary adder



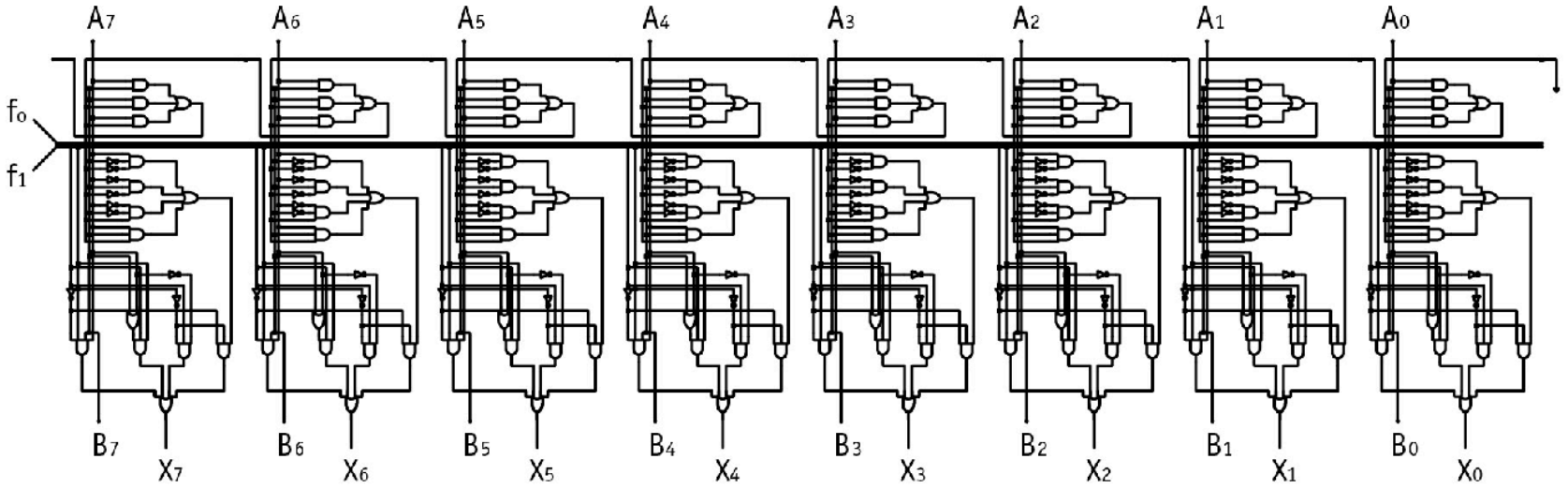
1 bit binary adder



26 transistors

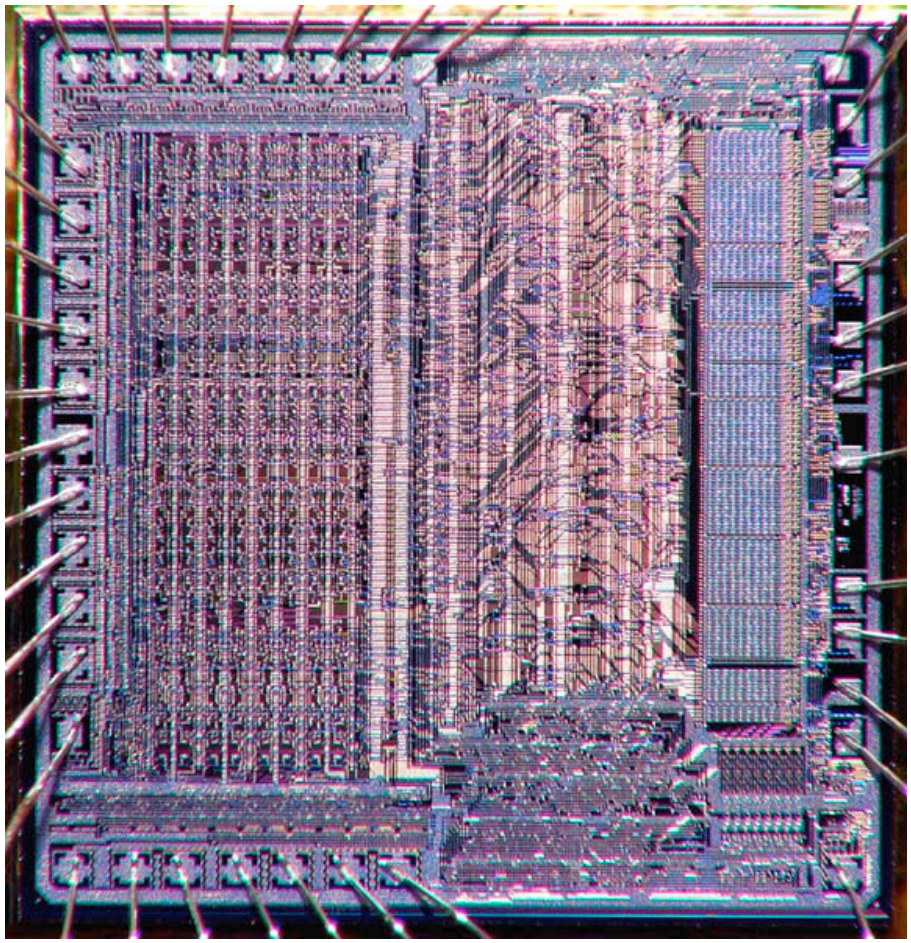
US Patent
4,831,578

8 bit binary adder



6800 CPU

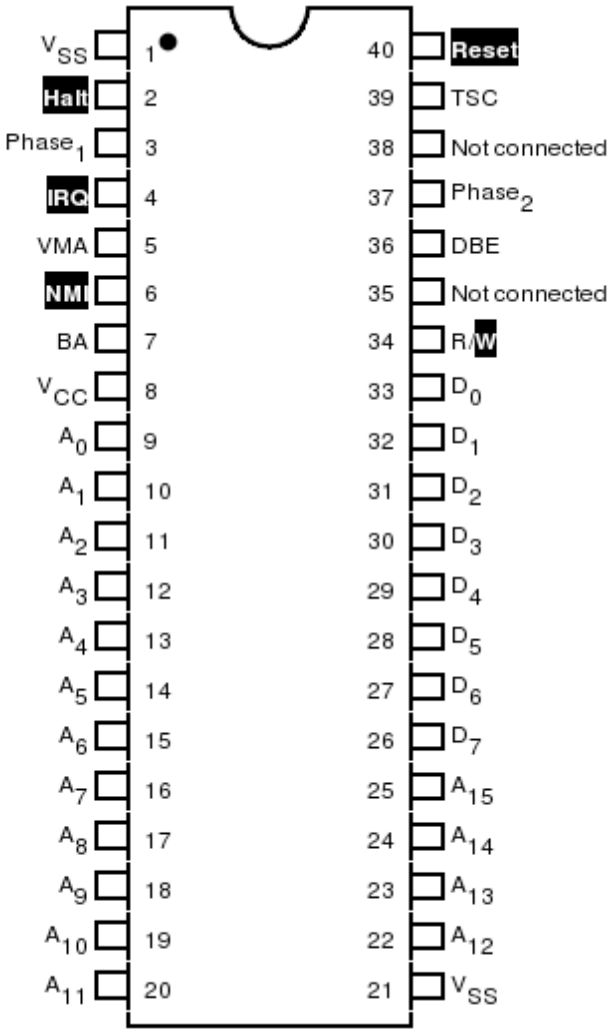
- Introduced 1974
- 4000 transistors
- 1.0-2.5 MHz
- 3, 8 bit registers
- 3, 16 bit registers



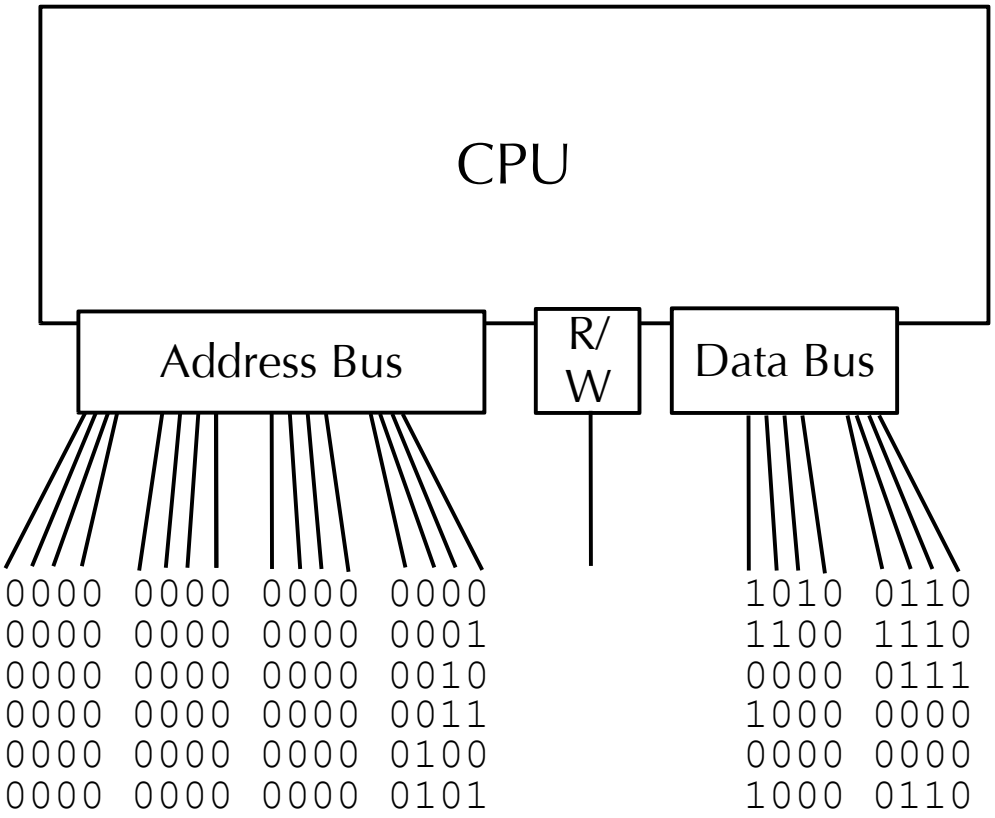
www.cpu-world.com

Motorola 6800 CPU

- 72 instructions (197 opcodes)
- 8 bit data bus (0-255)
- 16 bit address bus (64k max RAM)
- 6 registers:
  - 8 bit ACCA
  - 8 bit ACCB
  - 16 bit IX
  - 16 bit PC
  - 16 bit SP
  - 6 bit CC



CPU Communications

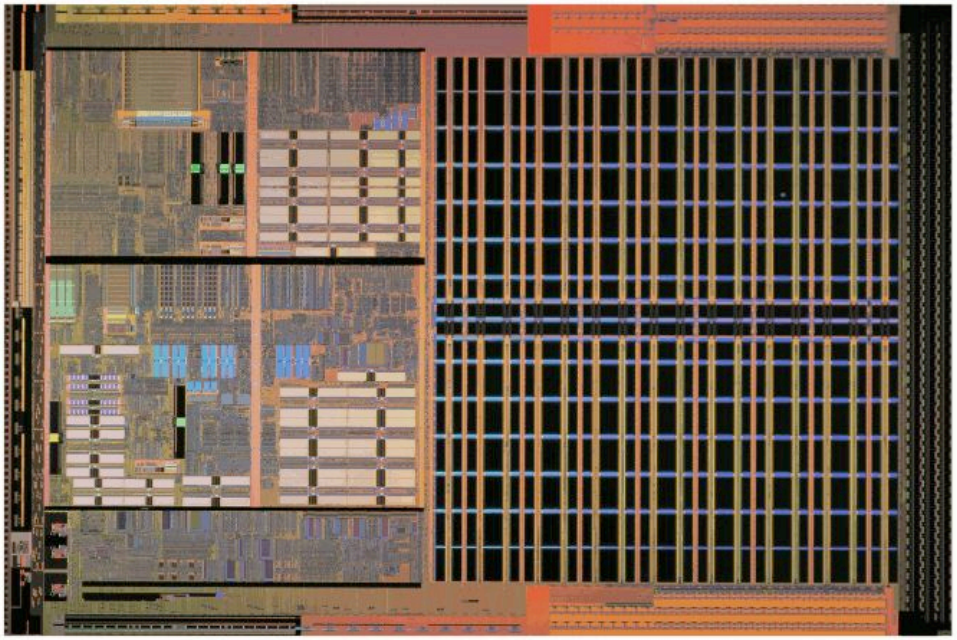


6800 Assembly

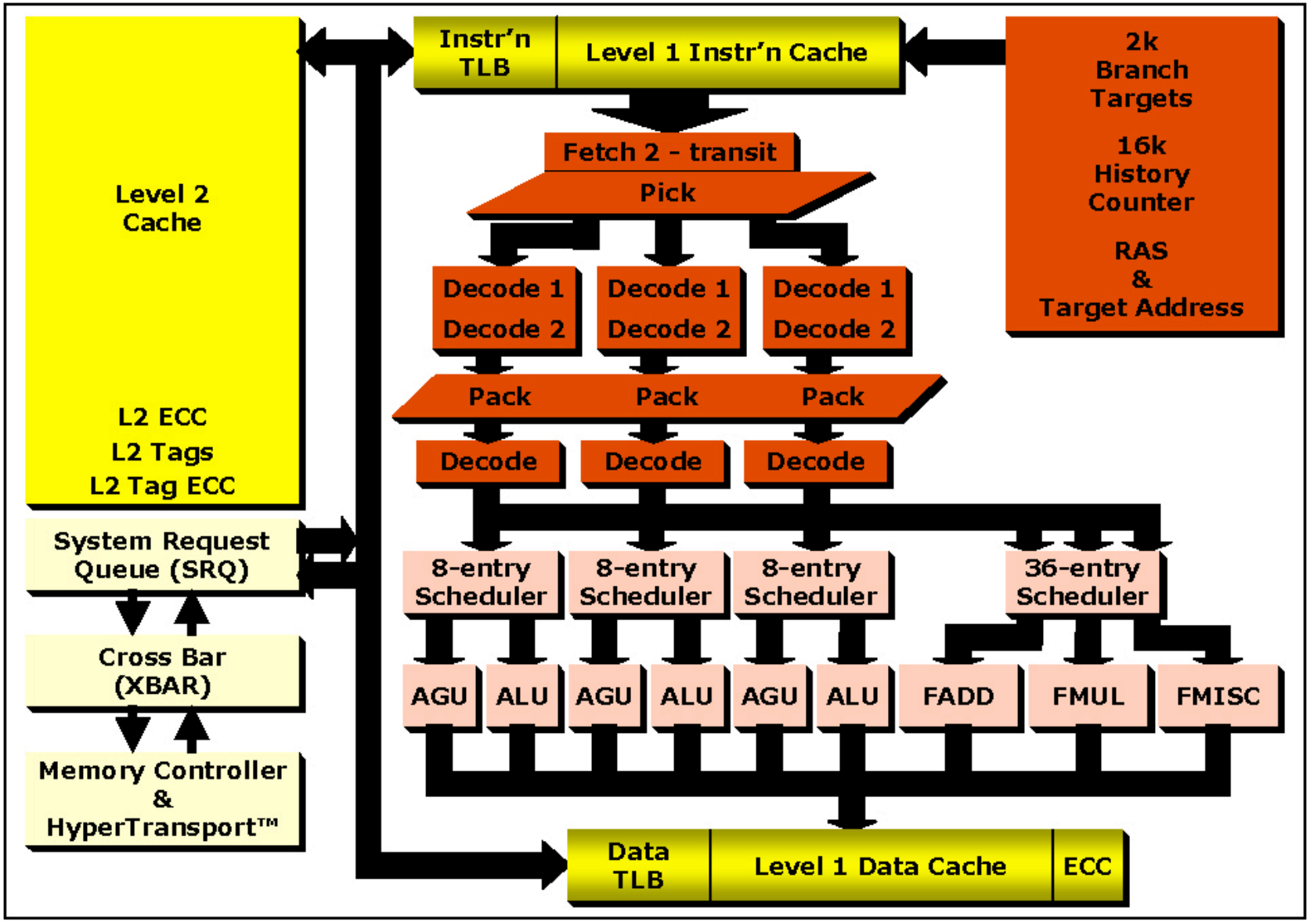
```
33+10/2

Memory:
0000: 1000 0110        LDA
0001: 0010 0001        33                  33 -> ACCA
0002: 1000 1011        ADDA
0003: 0000 1010        10                  43 -> ACCA
0004: 0100 0110        RORA                21 -> ACCA
0005: 1001 0111        STAA
0006: 0000 1010 10     ACCA -> mem(10)
...
000A: 0001 0101
```

Athlon-64

- ~106 million transistors (~10 m^3 if individually packaged)
- Socket-939 (939 pins)
- 40 bit addressing (1 TB)
- 64 bit data bus
- ~2 GHz
- registers:
- 16, 64 bit integer
- 16, 128 bit 'media'
- 8, 64 bit float



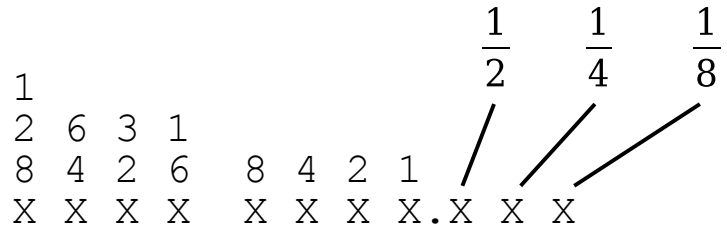
Opteron Execution Units



Microprocessors

- 6800 ~4000 transistors, 8 bit (1974)
- 68000 ~70,000 transistors, 16/32 bit (1979)
- 68040 ~1,200,000 transistors 32 bit+FPU (1990)
- Core2 duo ~291,000,000 transistors 64 bit (2006)

Binary Numbers




```
1           0000 0001
2           0000 0010
4           0000 0100
8           0000 1000
15          0000 1111
212         1101 0100

0.25        0000 0000.010
0.625       0000 0000.101
0.3         0000 0000.0100 1100 1100 ...
```

IEEE Floating Point

- Single (float) 1/24/8 bits 7 digits 10^{38}
- Double 1/53/11 bits 15 digits 10^{308}
- Long Double 1/64/16 bits 18 digits 10^{9864}

Single:

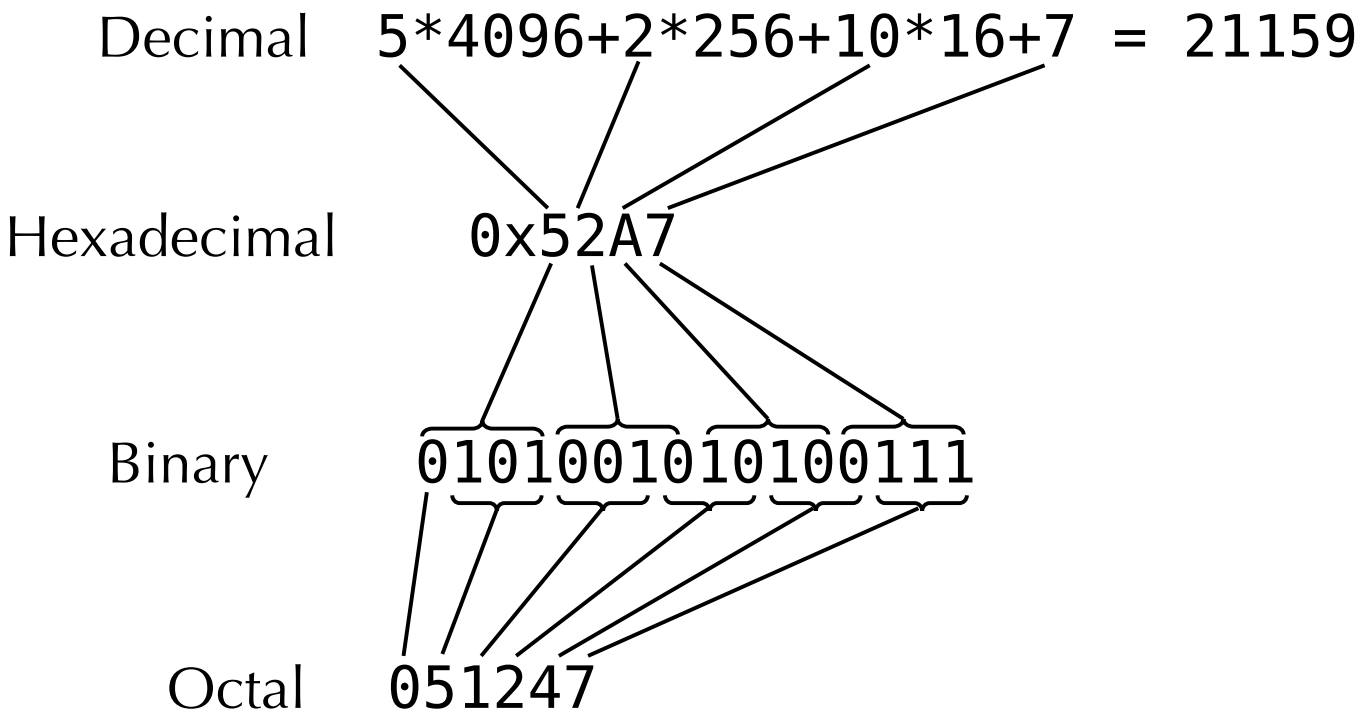
SEEEEEEE EMMMMMMM MMMMMMMM MMMMMMMM

S – Sign bit 0=+

E – Exponent, bias 127

M – Significand (Mantissa), implicit 1 when normalized

Bases


Decimal
5*4096+2*256+10*16+7 = 21159
Hexadecimal
0x52A7
Binary
0101001010100111
Octal
051247

Digital Representation of Numbers

• Bit		0-1
• Nibble	(4 bits)	0-15
• Byte (char)	(8 bits)	0-255
• Word (short)	(16 bits)	0-65,535
• Longword (long)	(32 bits)	0-4,294,967,296
• Long Longword	(64 bits)	$0\text{-}1.844 \times 10^{19}$
• Float	(32 bits)	10^{38}
• Double	(64 bits)	10^{308}

IEEE Floating Point

```
float f=0;
for (int i=1; i<1000; i++) f+=i;
printf("%f\n",f/999.0);

500.0
```

IEEE Floating Point

```
float f=0;
for (int i=1; i<1000000; i++) f+=i;
printf("%f\n",f/999999.0);
```

IEEE Floating Point

```
float f=0;
for (int i=1; i<1000000; i++) f+=i;
printf("%f\n",f/999999.0);

499940.86013
```

IEEE Floating Point

```
float f=0;
for (int i=1; i<1000000; i++) f+=i;
printf("%f\n",f/999999.0);

499940.86013

float f=0;
for (int i=999999; i>0; i--) f+=i;
printf("%f\n",f/999999.0);

499873.194145
```

IEEE Floating Point

```
double f=0;
for (int i=1; i<1000000; i++) f+=i;
printf("%f\n",f/999999.0);

500000.0

double f=0;
for (int i=999999; i>0; i--) f+=i;
printf("%f\n",f/999999.0);

500000.0
```

File Manipulation

- os.listdir - Lists files in a particular folder
- os.stat - info about a file
- os.rename - rename (mv) a file
- os.mkdir - create a folder
- os.remove - delete a file
- os.rmdir - remove a directory
- os.system - execute a command (mostly mac/linux)

Homework #4

- We are going to write our first useful program (scientifically speaking). We will be computing a histogram of a list of numbers.

  The input to the program will be the name of a file containing a list of numbers. Each line will contain a single number. Read this file in (you can use argv or prompt the user for a filename). Now produce a histogram of the numbers, that is, you have a set of (max-min)/bin_size 'bins', and you need to count how many of the numbers fall into each bin, then present the results. **Do not ask the user for the histogram parameters, but come up with intelligent values based on the data.** Max/min are not necessarily the max and min of the data, depending on how you decide to handle unusual cases. You could consider using 'outlier' bins. The results are just the center of each bin and how many numbers are in it.

  The class website will have 3 example files to download and test your program on. Do not change the program between the 3 runs. Your program should generate reasonable results for all 3. Please email your program and the results from running it on each of the 3 data sets.

  If you aren't feeling challenged, you could try to come up with a more 'artistic' way to present the results using text output.

- Install the Python Imaging Library (PIL) on your computer:

  http://www.pythonware.com/products/pil

  or 'easy_install pil'

  Test by doing an 'import PIL' in python