Computation

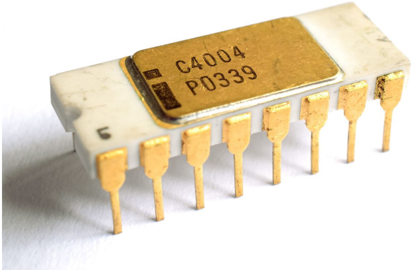
Abstraction and Implementation

Dominic Jones

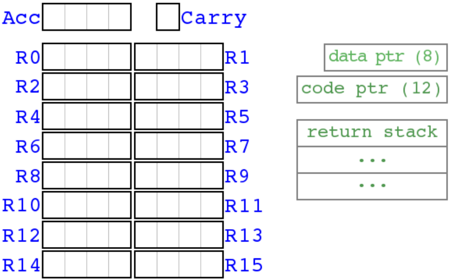
CPU - Logic - Transistor

Intel 4004

4-bit data, 2,300 transistors



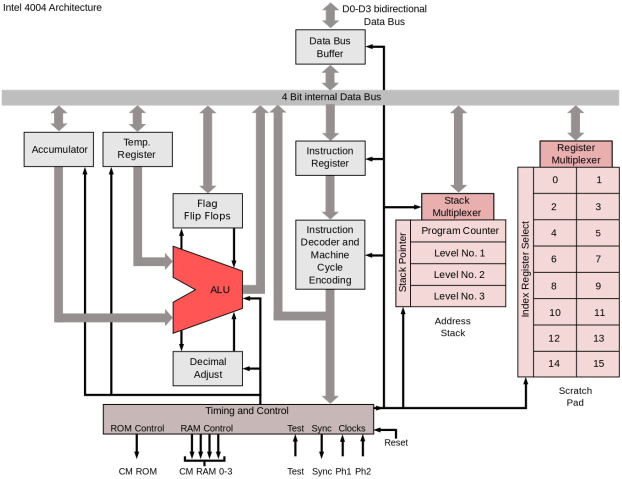
16 general registers



Instruction set contains 46 instructions (3,683 in x86-64)

Intel 4004

Single cycle data path architecture



Intel 4004 Emulator

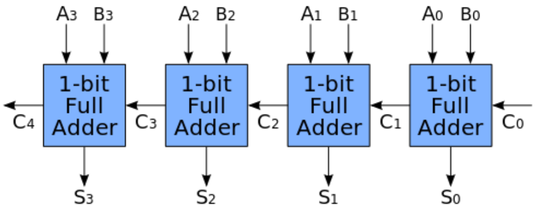
59 + 38 = 91

```
; sum lower digits (9,8)
clc      ; car = 0
ld   r1  ; acc = r1
add  r3  ; acc = acc + car + r3
xch  r1  ; r1  = acc

; sum higher digits (5,3)
ld   r0  ; acc = r0
add  r2  ; acc = acc + car + r2
xch  r0  ; r0  = acc
```

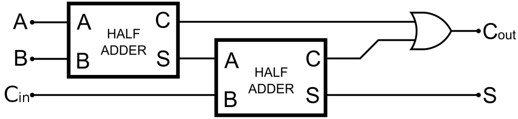
acc	car	r0	r1	r2	r3
x	x	5	9	3	8
x	0	5	9	3	8
9	0	5	9	3	8
1	1	5	9	3	8
1	1	5	1	3	8
5	1	5	9	3	8
9	0	5	9	3	8
9	0	9	1	3	8

Instructions

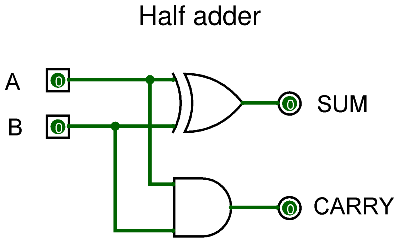
4-bit ripple carry adder

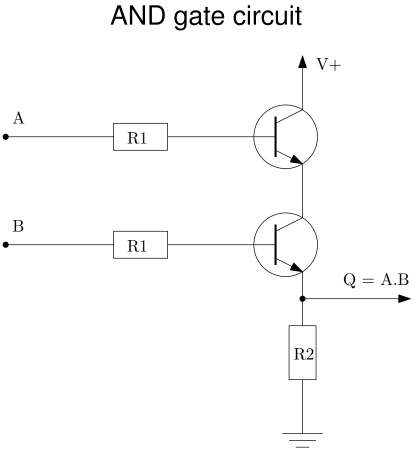


1-bit full adder



Transistors





Mathematics - Computation

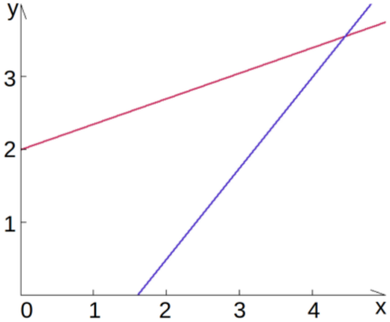
Intersection of two lines

$$\begin{aligned} -0.35x + y &= 2 \\ 2.5x - 2y &= 4 \end{aligned}$$

$$\begin{bmatrix} -0.35 & 1 \\ 2.5 & -2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -0.35 & 1 \\ 2.5 & -2 \end{bmatrix}^{-1} \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

Matrix inverse?



Division (scalar inverse)

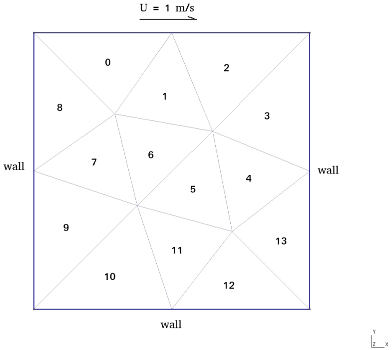
```
unsigned divide(unsigned a, unsigned b)
{
  unsigned denom = b, current = 1, result = 0;

  if (denom > a) return 0;
  if (denom == a) return 1;

  while (denom <= a) {
    denom <<= 1;
    current <<= 1;
  }

  denom >>= 1;
  current >>= 1;

  while (current != 0) {  // n^1
    if (a >= denom) {
      a -= denom;
      result |= current;
    }

    current >>= 1;
    denom >>= 1;
  }

  return result;
}
```

Matrix inverse

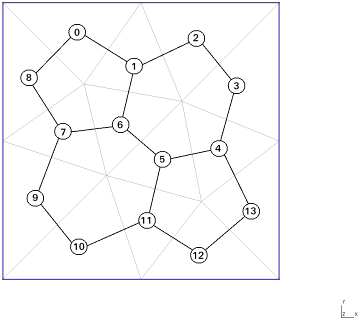
```
void inverse(int n, float a[][2*n_max])
{
  for (int i = 0; i < n; i++)
    for (int j = n; j < 2*n; j++)
      a[i][j] = (i == j-n? 1: 0);

  for (int i = 0; i < n; i++) {          // n^1
    float aii = a[i][i];
    for (int j = i; j < 2*n; j++)
      a[i][j] = a[i][j] / aii;

    for (int j = 0; j < n; j++) {        // n^2
      if (i != j) {
        float aji = a[j][i];
        for (int k = 0; k < 2*n; k++)    // n^3
          a[j][k] = a[j][k] - aji * a[i][k];
      }
    }
  }
}
```

Simulation: cavity flow

Geometry and mesh



Connectivity graph



Topology Representation

Symmetric, sparse (48 non-zeros), irregular

$$\begin{bmatrix}
a_{0,0} & a_{0,1} & & & & & & & a_{0,8} & & & & & \\
a_{1,0} & a_{1,1} & a_{1,2} & & & & a_{1,6} & & & & & & & \\
& a_{2,1} & a_{2,2} & a_{2,3} & & & & & & & & & & \\
& & a_{3,2} & a_{3,3} & a_{3,4} & & & & & & & & & \\
& & & a_{4,3} & a_{4,4} & a_{4,5} & & & & & & & & a_{4,13} \\
& & & & a_{5,4} & a_{5,5} & a_{5,6} & & & & & a_{5,11} & & \\
& a_{6,1} & & & & a_{6,5} & a_{6,6} & a_{6,7} & & & & & & \\
& & & & & & a_{7,6} & a_{7,7} & a_{7,8} & a_{7,9} & & & & \\
a_{8,0} & & & & & & & a_{8,7} & a_{8,8} & & & & & \\
& & & & & & & a_{9,7} & & a_{9,9} & a_{9,10} & & & \\
& & & & & & & & & a_{10,9} & a_{10,10} & a_{10,11} & & \\
& & & & & a_{11,5} & & & & & a_{11,10} & a_{11,11} & a_{11,12} & \\
& & & & & & & & & & & a_{12,11} & a_{12,12} & a_{12,13} \\
& & & & a_{13,4} & & & & & & & & a_{13,12} & a_{13,13}
\end{bmatrix}$$

Sparsity and Indirection

- Dense storage: 196 values, 24% efficient
- Direct access to values

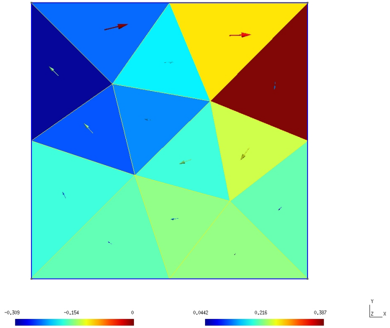
```
float[13][13] A;
A[2][3] = 3.142;
```

- Compressed row storage: 111 values, 56% efficient
- Requires indirection to access values (10x slower)

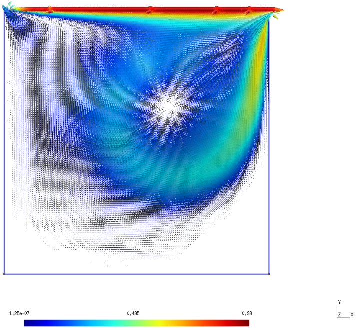
```
int[15] IA = [0, 3, 7, 10, ...];
int[48] JA = [0, 1, 8,  0, 1, 2, 6,  1, 2, 3,  ...];
float[48] A;

assert(JA[IA[2]+2] == 3);

A[JA[IA[2]+2] = 3.142; // i.e. A[2][3] = 3.142;
```

Accuracy

Coarse mesh (48 elements)

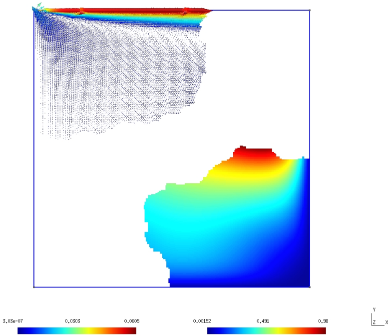


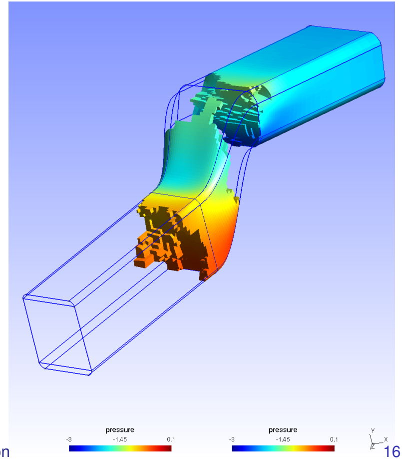
Fine mesh (132,607 elements)



Speed

1. Partition the mesh into four chunks
2. Each CPU core computes one chunk
3. Communication eventually swamps computation

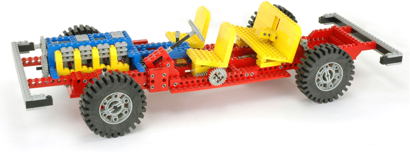




Fundamental Tensions (by analogy)

Complexity - Proliferation

[1977] 601 parts, 99 part types



[2016] 3929 parts, 3147 part types

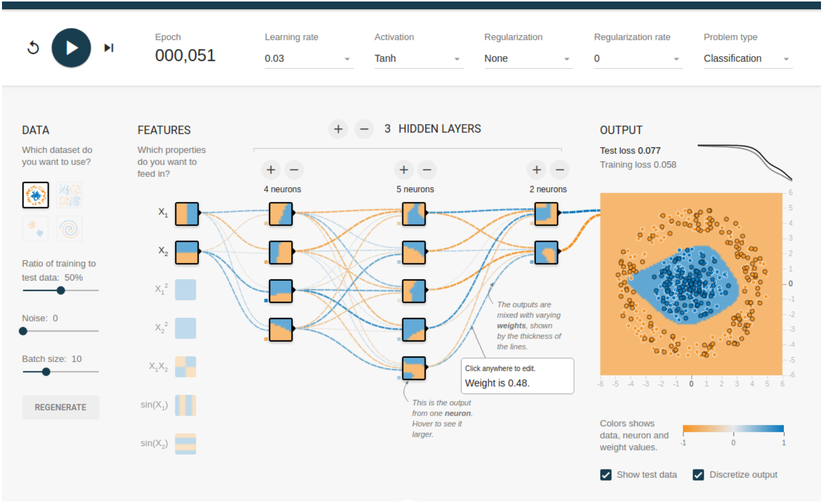
Indirection - Efficiency

[1989] Pneumatics
flexible, weak

[2007] Linear actuators
powerful, bulky

Artificial Neural Networks - Machine Learning

TensorFlow demo



From the web

1. Intel 4004 Python emulator
2. Logisim circuit simulation
3. Bluebit matrix calculator
4. Gmsh meshing and post-processing
5. Flowlab simulation code
6. TensorFlow demo