<table>
<tr><th>Opcode</th><th>Description</th><th>Pseudo - Implementation</th></tr>
<tr><td>0NNN</td><td>Calls RCA 1802 program at address NNN. Not necessary for most ROMs.</td><td></td></tr>
<tr><td>00E0</td><td>Clears the screen.</td><td>Set’s all pixels from to 0</td></tr>
<tr><td>00EE</td><td>Returns from a subroutine.</td><td>Return to address stored in stack</td></tr>
<tr><td>1NNN</td><td>Jumps to address NNN.</td><td>Load NNN into PC</td></tr>
<tr><td>2NNN</td><td>Calls subroutine at NNN.</td><td>Store PC in stack then load NNN into PC</td></tr>
<tr><td>3XNN</td><td>Skips the next instruction if VX equals NN.</td><td>If V[X] == NN then increment PC by 4 (this instruction + next)</td></tr>
<tr><td>4XNN</td><td>Skips the next instruction if VX doesn't equal NN.</td><td>If V[X] != NN then increment PC by 4 (this instruction + next)</td></tr>
<tr><td>5XY0</td><td>Skips the next instruction if VX equals VY.</td><td>If V[X] == V[Y] then increment PC by 4 (this instruction + next)</td></tr>
<tr><td>6XNN</td><td>Sets VX to NN.</td><td>Set V[X] = NN</td></tr>
<tr><td>7XNN</td><td>Adds NN to VX.</td><td>V[X] += NN</td></tr>
<tr><td>8XY0</td><td>Sets VX to the value of VY.</td><td>V[X] = V[Y]</td></tr>
<tr><td>8XY1</td><td>Sets VX to VX or VY.</td><td>V[X] = V[X] | V[Y]</td></tr>
<tr><td>8XY2</td><td>Sets VX to VX and VY.</td><td>V[X] = V[X] & V[Y]</td></tr>
<tr><td>8XY3</td><td>Sets VX to VX xor VY.</td><td>V[X] = V[X] ^ V[Y]</td></tr>
<tr><td>8XY4</td><td>Adds VY to VX. VF is set to 1 when there's a carry, and to 0 when there isn't.</td><td>if((V[X] + V[Y]) > 255){
V[F] = 1;
V[X] += V[Y];
}
Else{
V[X] += V[Y];
}</td></tr>
<tr><td>8XY5</td><td>VY is subtracted from VX. VF is set to 0 when there's a borrow, and 1 when there isn't.</td><td>if(V[X] - V[Y] < 0){
V[X] -= V[Y]
V[F] = 0
}
Else{
V[X] -= V[Y];
}</td></tr>
<tr><td>8XY6</td><td>Shifts VX right by one. VF is set to the value of the least significant bit of VX before the shift.</td><td>V[F] = V[X] & 1; //AND rightmost bit
V[X] >> 1;</td></tr>
<tr><td>8XY7</td><td>Sets VX to VY minus VX. VF is set to 0 when there's a borrow, and 1 when there isn't.</td><td>if(V[Y] - V[X] < 0){
V[F] = 0;
V[X] = V[Y] - V[X];
}
Else{</td></tr>
</table>

		V[X] = V[Y] - V[X]; }
8XYE	Shifts VX left by one. VF is set to the value of the most significant bit of VX before the shift.	V[F] = V[X] & 10000000 V[X] << 1
9XY0	Skips the next instruction if VX doesn't equal VY.	V[X] != V[Y]
ANNN	Sets I to the address NNN.	I = NNN
BNNN	Jumps to the address NNN plus V0.	Pc = NNN + v[0]
CXNN	Sets VX to the result of a bitwise and operation on a random number and NN.	V[X] = NN & (rand() % 0xFF)

DXYN	Draws a sprite at coordinate (VX, VY) that has a width of 8 pixels and a height of N pixels. Each row of 8 pixels is read as bit-coded starting from memory location I; I value doesn't change after the execution of this instruction. As described above, VF is set to 1 if any screen pixels are flipped from set to unset when the sprite is drawn, and to 0 if that doesn't happen	//http://www.multigesture.net/articles/how-to-write-an-emulator-chip-8-interpreter/
		
		unsigned short x = V[(opcode & 0x0F00) >> 8];
		unsigned short y = V[(opcode & 0x00F0) >> 4];
		unsigned short height = opcode & 0x000F;
		unsigned short pixel;
		
		V[0xF] = 0;
		for (int yline = 0; yline < height; yline++)
		{
		pixel = memory[I + yline];
		for(int xline = 0; xline < 8; xline++)
		{
		if((pixel & (0x80 >> xline)) != 0)
		{
		if(gfx[(x + xline + ((y + yline) * 64))] == 1)
		V[0xF] = 1;
		gfx[x + xline + ((y + yline) * 64)] ^= 1;
		}
		}
		}
		
		drawFlag = true;
		pc += 2;

EX9E	Skips the next instruction if the key stored in VX is pressed.	If Key == V[X] PC += 4
EXA1	Skips the next instruction if the key stored in VX isn't pressed.	If key != V[X] PC += 4
FX07	Sets VX to the value of the delay timer.	V[X] = delay_timer
FX0A	A key press is awaited, and then stored in VX.	V[X] = key
FX15	Sets the delay timer to VX.	Delay_timer = V[X]
FX18	Sets the sound timer to VX.	Sound_timer = V[X]
FX1E	Adds VX to I.	I += V[X]

FX29	Sets I to the location in memory of the sprite for the character in VX. Characters 0-F (in hexadecimal) are represented by a 4x5 font.	//http://laurencescotford.co.uk/?p=440
		I = 0x8100 | (memory[0x8100 | (V[X] & 0x0F)])

FX33	Stores the binary-coded decimal representation of VX, with the most significant of three digits at the address in I, the middle digit at I plus 1, and the least significant digit at I plus 2. (In other words, take the decimal representation of VX, place the hundreds digit in memory at location in I, the tens digit at location I+1, and the ones digit at location I+2.)	memory[I] = V[(opcode & 0x0F00) >> 8] / 100;
		
		memory[I + 1] = (V[(opcode & 0x0F00) >> 8] / 10) % 10;
		
		memory[I + 2] = (V[(opcode & 0x0F00) >> 8] % 100) % 10;
		
		pc += 2;

FX55	Stores V0 to VX (including VX) in memory starting at address I.	For n = 0; n < F; n++ memory [I + n] = V[n]
FX65	Fills V0 to VX (including VX) with values from memory starting at address I.	For n = 0; n < F; n++ V[n] = memory[I + n]