The SOS machine is made up of a number of units. Memory consists of 4096 32-bit words which contain either instructions or data. The words are numbered in hex from 000 to FFF. They are not in two's-complement form; all addresses are positive. The first 16 words (000-0FF) are called general registers. These registers can function as any other word of memory, but they are faster, and are used for other purposes as well.

The memory select is a "switchboard" which makes a connection between any word of memory and the CPU. Included in the CPU are the instruction counter (IC) which tells which instruction to execute next and the instruction register (IR) which contains a copy of the instruction currently being executed.

The remaining unit is the adder (and adder select) which is a special device for adding 2 32 bit numbers.

Processing an instruction
The complete cycle for carrying out 1 instruction consists of 3 parts: fetch/increment, parse and execution:

1. The CPU looks in the IC and gets the address of the instruction to be executed.
2. The CPU sends that address to the memory select and the corresponding word of memory is put in the IR.
3. The instruction counter is incremented by 1 for the next instruction.

Parse
4. Now the CPU "takes apart" the instruction in the IR and figures out what to do. The instruction (8 hex digits) is broken up like so:

<table>
<thead>
<tr>
<th>O</th>
<th>P</th>
<th>CODE</th>
<th>REG</th>
<th>X</th>
<th>I</th>
<th>ADDR</th>
</tr>
</thead>
</table>

Where do | What to do | I From | Where
---|---|---|---|---|---|---|
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
5. Finally, the instruction is carried out. This is best illustrated by an example. If the instruction is 10400200 (add contents of loc. 100 to P4), then it is broken up as:

\[
\begin{array}{cccc}
\text{10} & \text{4} & \text{0} & \text{0} \\
\text{OP CODE} & \text{REG.} & \text{FORGET ABOUT} & \text{ADDRESS} \\
\text{(ADD)} & \text{THESE} & \\
\end{array}
\]

Then the following steps are done:
(a) Reset memory select to 000.
(b) Reset adder select to top adder.
(c) Move from memory to adder.
(d) Reset memory select to 100.
(e) Reset adder select to middle adder.
(f) Move memory to adder.
(g) Do the addition.
(h) Reset memory select to 000.
(i) Reset adder select to bottom adder.
(j) Move from adder to memory.

**Addressing**

The effective address is the memory address in which the data for an instruction is found. In the previous instruction the effective address was 100. Sometimes the calculation of the effective address is more complicated than in the example.

Digit number 3 of the instruction specifies the index register. This can be any of the the registers 0-F (the first 16 words of memory). Then calculating an indexed address you add the contents of the index register to the last 3 digits of the instruction to get the effective address. The only exception to this is specifying an index register of 0 which means don't index on this instruction.

Indexing is useful for looping. If for instance we wanted to add the contents of memory location 100-153 we could use the following set of instructions:

\[
\begin{align*}
R3 & \leftarrow 0 \quad \text{(PUT A ZERO IN R3)} \\
R4 & \leftarrow R4 + 100(R3) \quad \text{(ADD TO R4 THE CONTENTS OF 100 INDEXED BY R3)} \\
R3 & \leftarrow R3 + 1
\end{align*}
\]

The instruction \( R4 \leftarrow R4 + 100(R3) \) would be written 10430100.

A further complication to calculating the effective address is indirect addressing. If digit 4 of the instruction is odd (the last bit is on) then the word at the effective address (using indexing if necessary) is to be used to calculate a new effective address, which will contain the data. So the instruction 104001100
means use the word at location 180 to find out what it is to be added to R8. If location 180 contains xxx20074 (we ignore the first 3 digits) then location 274 contains the data to add to R8. If, however, 180 contained xxx32274 then we would have to use indirect addressing again to find the data.

The Loader

The loader is a program which takes 650 machine instructions (or data) off of cards and puts them in consecutive words of memory. After the last card, insert a card with the pseudonamespace FFF (loc) which says: stop loading and begin executing at (loc).

More on this soon.