Relocatable Code

Relative Jumps

The big idea about relocatable code is to simply have a relative jump operation. I.E. the operation itself is coded relative. That's not a big issue but the question then is if you are writing code to be relocatable, why ever use a normal jump? You would need at minimum bra (branch), lbra (long branch) and jmpr for 1, 2 and 3 byte encodings. So many jumps! Is this really the way? Well, it would work instantly, but in this case why not just make all jumps relative? Well, you would still need absolute jumps.

As we can see the issue with adding relative jump commands is that it muddies the ISA. In the end you will have two versions of each jump command.

Code Analysis

No code analysis will be a simple solution because it is extremely hard to 'scan' for jump operations. If the code was 4-aligned, maybe, but this is not the case here.

Relative Flag

The other idea I had is to set a CPU flag that would treat any jump as a relative jump. The question then becomes how to translate all the jumps from absolute to relative? What this ends up meaning is that the CPU needs to know – at any time – the start address of the program. This lends some credibility to the INTERRUPT flag which would have to turn off relative mode.

The first idea I had was a relative flag that treated a JMP as relative. But now I think maybe this is more of a CPU thing in the sense that the CPU should have a start address register of sorts.

MMU address translation

This is a step towards paging memory management.

All we need is a special register – maybe, BP or BR for base pointer or register. When this address is 0, the system works as normal. However, when this address is not 0, AND the INTERRUPT_FLAG is not set (i.e. we're in normal code) the CPU will ADD the address in BP to any JMP operation. Which means, programs can be assembled into absolute space and then moved. All we need then is the difference between the program's load address and the space we are loading it into:

```
  BP = load space - load address
```

So a program being loaded at 0 will receive:

```
BP = 0 - $C000
BP = $FF4000
```

Alternately we could keep it simple by calling it the SALAD system. We have SA and LA. Start Address and Load Address. When the RF (RELATIVE_FLAG) is set, maybe, it uses SA and LA to calculate the address of any call, jump, load, etc.

Frankly this would be a pain to implement but it would be possible.