processor.cfg

```
// 0 -> OoO || 1 -> Superscalar In-order
SUPERSCALAR_IN_ORDER = 1;
```

Se ativo, IN_ORDER precisa ser 0!
No fim, esse é menos restritivo que o IN_ORDER.

Coloca uma dependência entre cada instrução e sua próxima.

- Isso é feito no estágio de **decode**, adicionando uma leitura à primeira uop e uma escrita à última
- Basicamente é adicionado um registrador de link o que é utilizado para dependências entre uops)
  - UOP_LINK_REGISTER = RAT_SIZE - 1 (Register to link uops)

processor_t

```
bool SUPERSCALAR_IN_ORDER;

INSTANTIATE_GET_SET(bool, SUPERSCALAR_IN_ORDER)
```

processor_t::allocate

```
...
if (cfg_processor.exists("SUPERSCALAR_IN_ORDER"))
     {
       set_SUPERSCALAR_IN_ORDER(cfg_processor["SUPERSCALAR_IN_ORDER"] ? true : false);
       assert (!IN_ORDER || !SUPERSCALAR_IN_ORDER);
     }
else
     { set_IN_ORDER(false); }
...
```

processor_t::decode

```
Logo antes de cada:
statusInsert = this->decodeBuffer.push_back(new_uop);


Adicionar:
// ******************************************************
// SUPERSCALAR IN ORDER processor
if (SUPERSCALAR_IN_ORDER) {
  if (uops_created == 1) {
    for (uint32_t i = 0; i < MAX_REGISTERS; i++)
    {
      if (new_uop.read_regs[i] == POSITION_FAIL)
      {
        new_uop.read_regs[i] = UOPS_LINK_REGISTER;
        break;
      }

    }
  } else if (uops_created == num_uops) {
    for (uint32_t i = 0; i < MAX_REGISTERS; i++)
    {
      if (new_uop.write_regs[i] == POSITION_FAIL)
      {
        new_uop.write_regs[i] = UOPS_LINK_REGISTER;
        break;
      }

    }
  }
}
// ******************************************************
```