What was it like in 1960?

- IBM, DEC, Univac
- Cap transistors
- Tape I/O, punch cards, all that
- FORTRAN, Algor, Lisp, Assembly (machine code)

OOO – Why bother?

- Efficiency
- Different latency of operations
- Was not significant
- No good compilers

Secret Decoder Ring

- Fi – output register
- Fj – input register 1
- Fk – input register 2
- Qj – functional unit producing operand j
- Qk – functional unit producing operand k
- Rj – ready flags
- Rk – ready flag
- Xi – per register reservation tag identifying functional unit that will produce its result.

Conditions Mem -> Issue

- Wait For
  - Wait for a free functional unit (of correct type)
  - Need the output register to not be an output for any active instruction.
- Do
  - Reserve the functional unit
  - Reserve the output register
  - If register is available, read it,
  - If register is not available, register interest it

Conditions Issue -> Read

- Waitfor
  - All operands to be available
- Do
  - Unregister our reservation on the operands

Read -> Execute

- Let it fly

Execute -> Complete

- WaitFor
  - End of latency on functional unit mem/etc
  - No one cares about the *old* output register value
- Do
  - Write the register
  - Update Xi flags
  - Un-reserve our functional unit
  - Trigger waiting instructions

Complete

When does this work?

- Independence
  - MUL R10, R11 -> R1
  - ADD R1, R2 -> R3
  - ADD R4, R5 -> R6

When does this not improve performance?

- Slim DFG
  - ADD r1, r2, r3
  - ADD r3, r4, r5
  - ADD r5, r6, r7
- Fewer registers create more conflicts in the output registers
  - ADD r1, r3, r4
  - ADD r4, r2, r5
  - ADD r7, r8, r4

Make this faster

```
MUL     R1, R2, R3
SUB     R10, R1, R13
ADD     R1, R5, R6
```

Make this faster

-
  ```
  MUL     R1, R2, R3
  ADD     R4, R5, R1
  ```

Make this faster

-
  ```
  MUL     R1, R2, R3
  MUL     R15, R16, R17
  SUB     R7, R8, R15
  ADD     R1, R5, R6
  ```