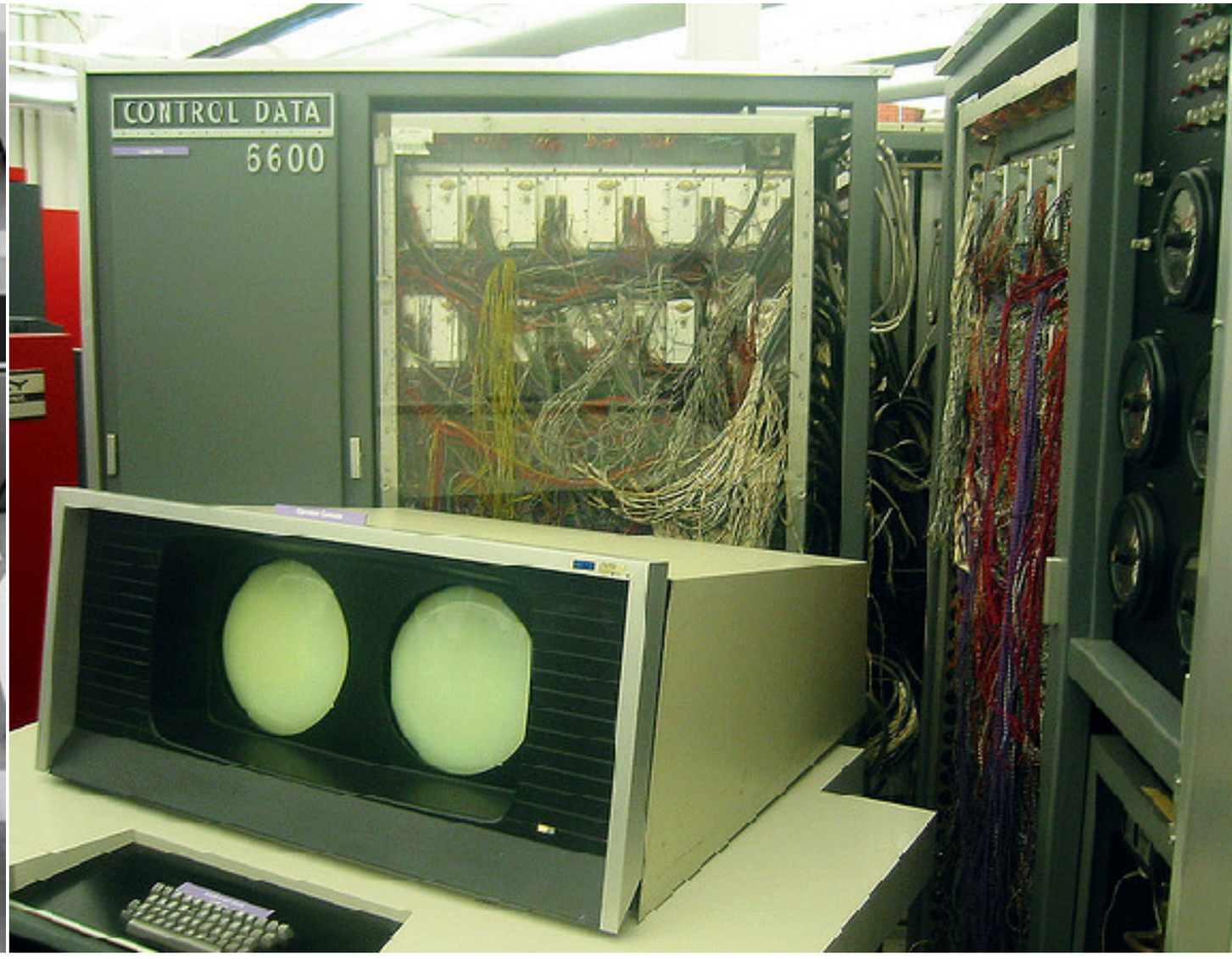
CDC 6600


CONTROL DATA
6600

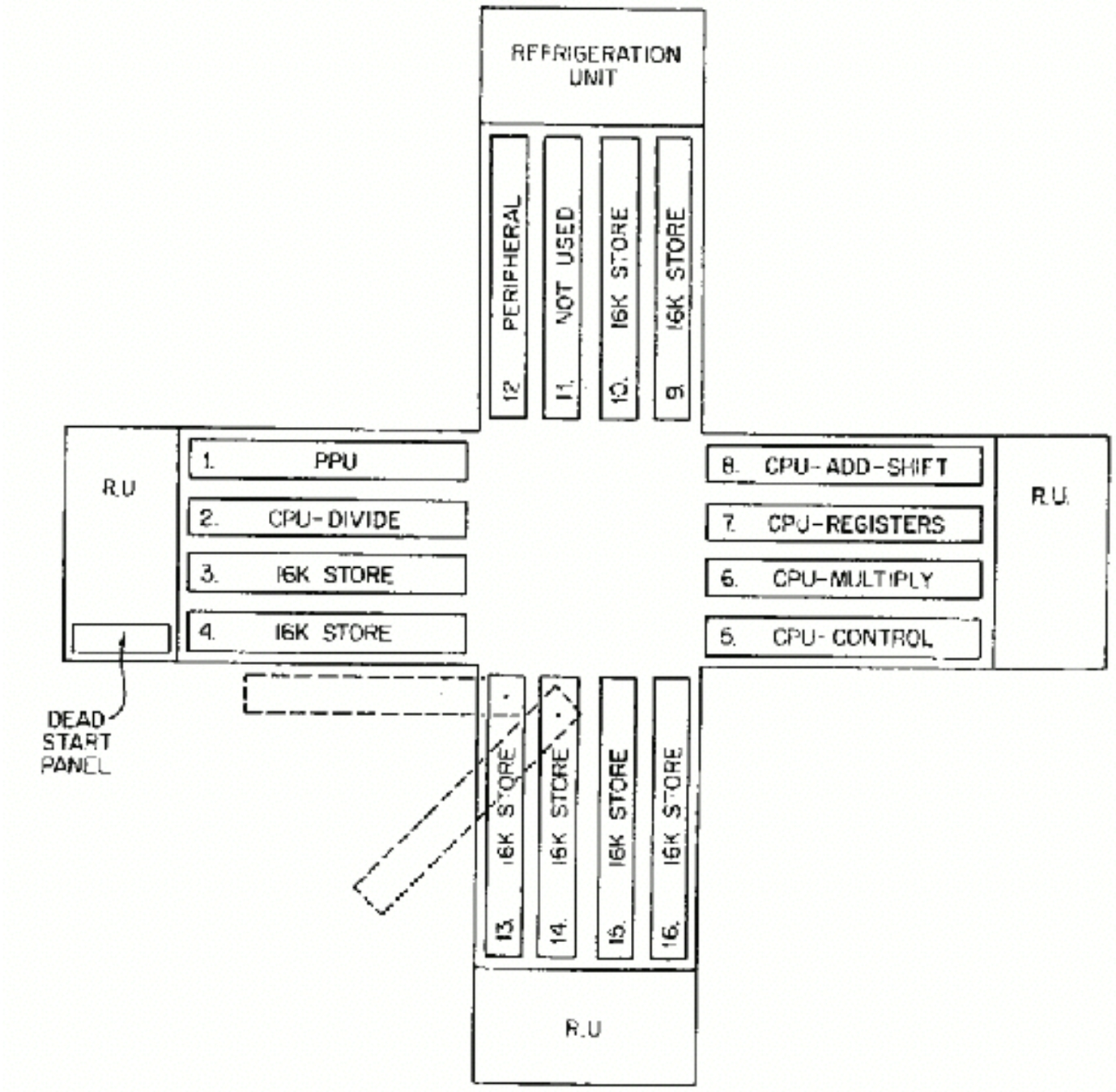
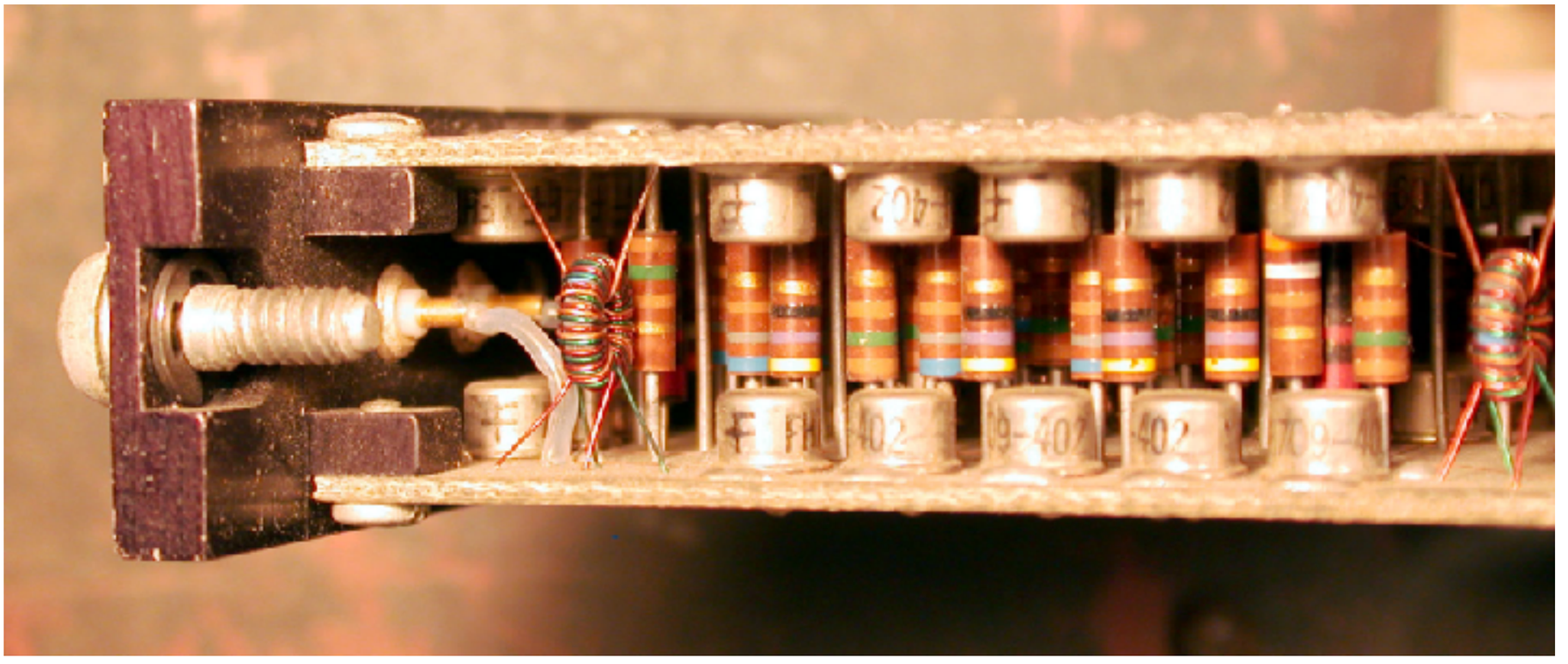
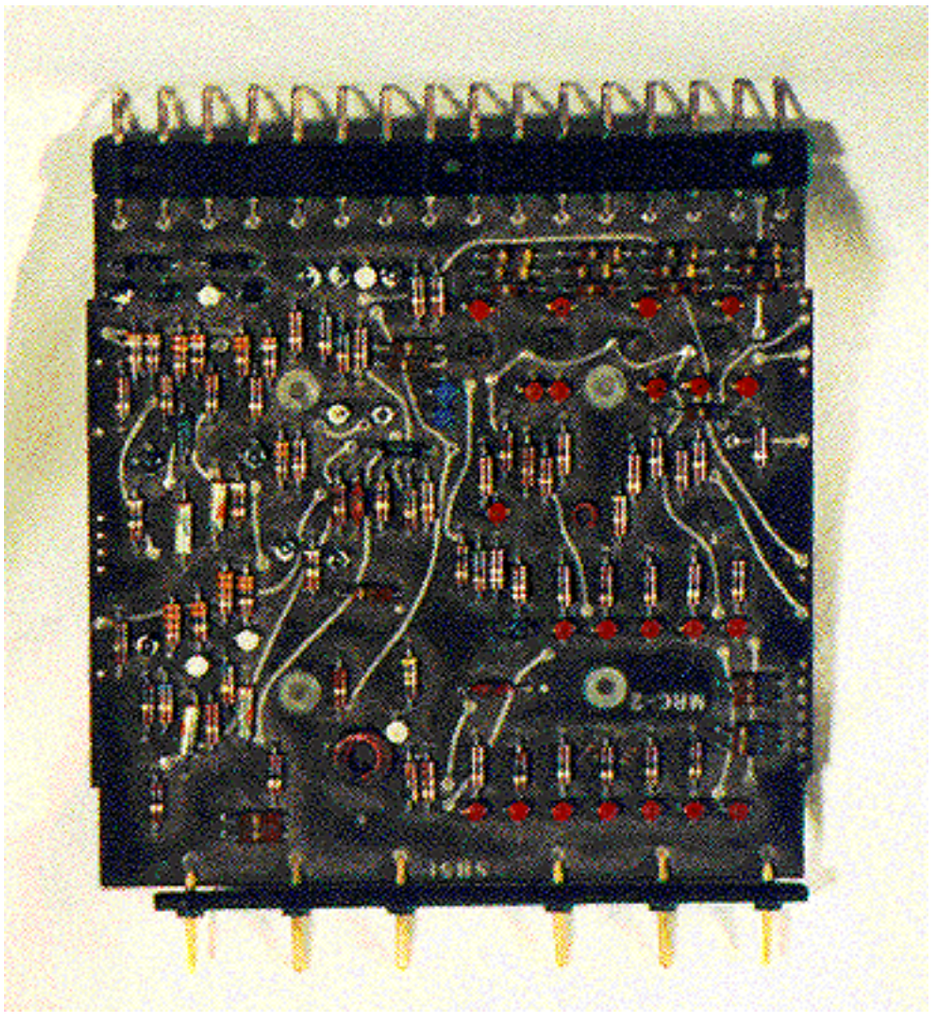
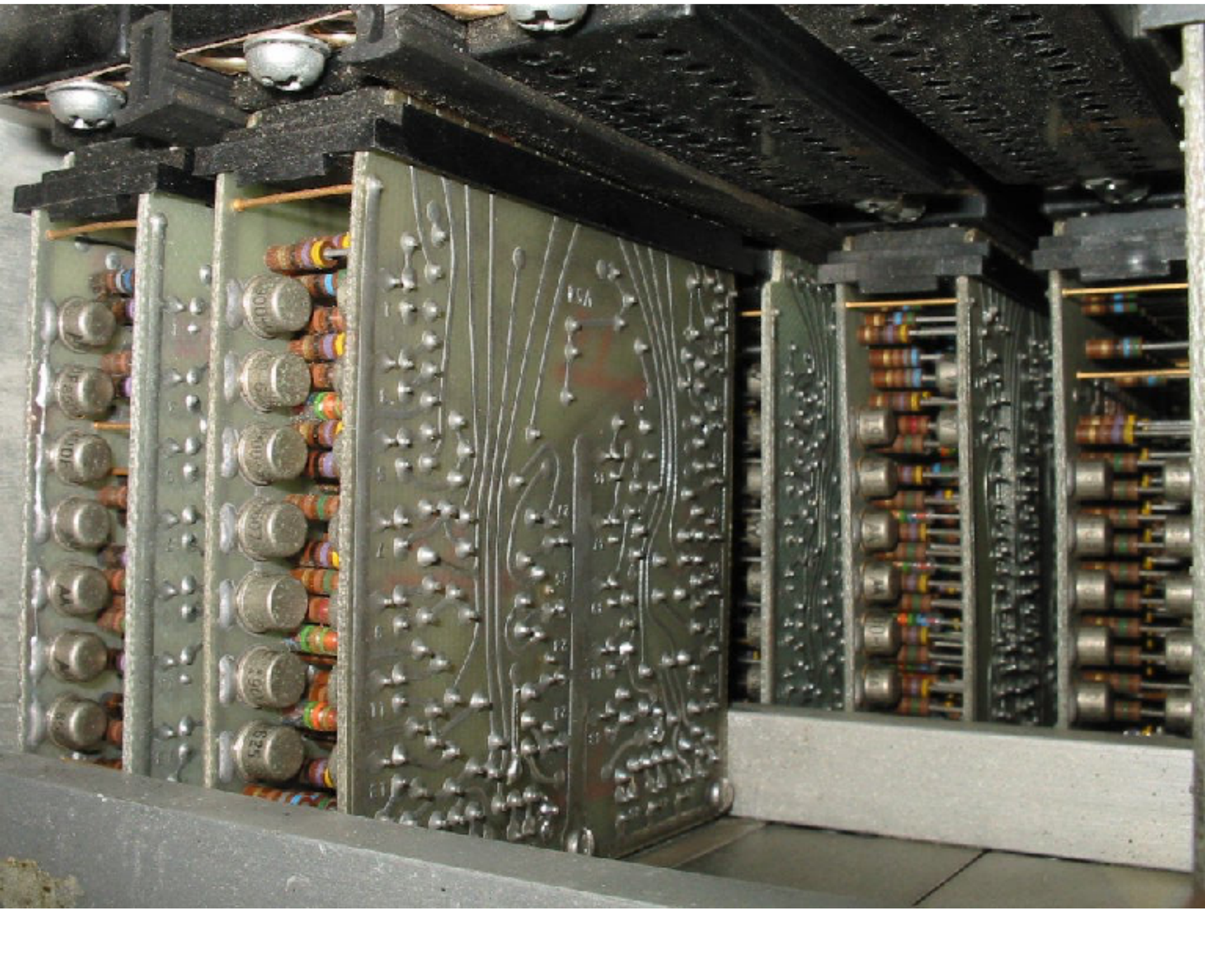


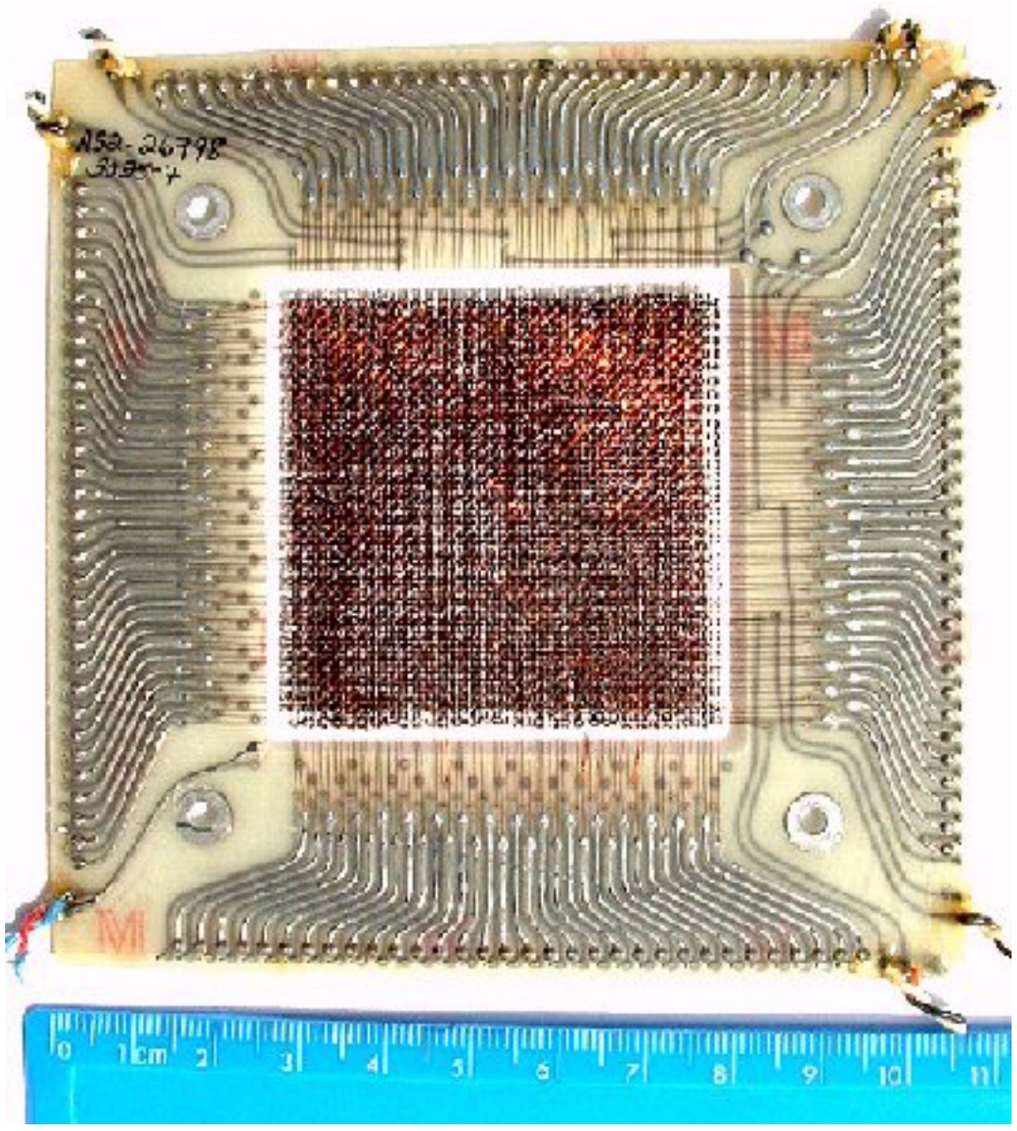
FIGURE 26 Cabinet top view.

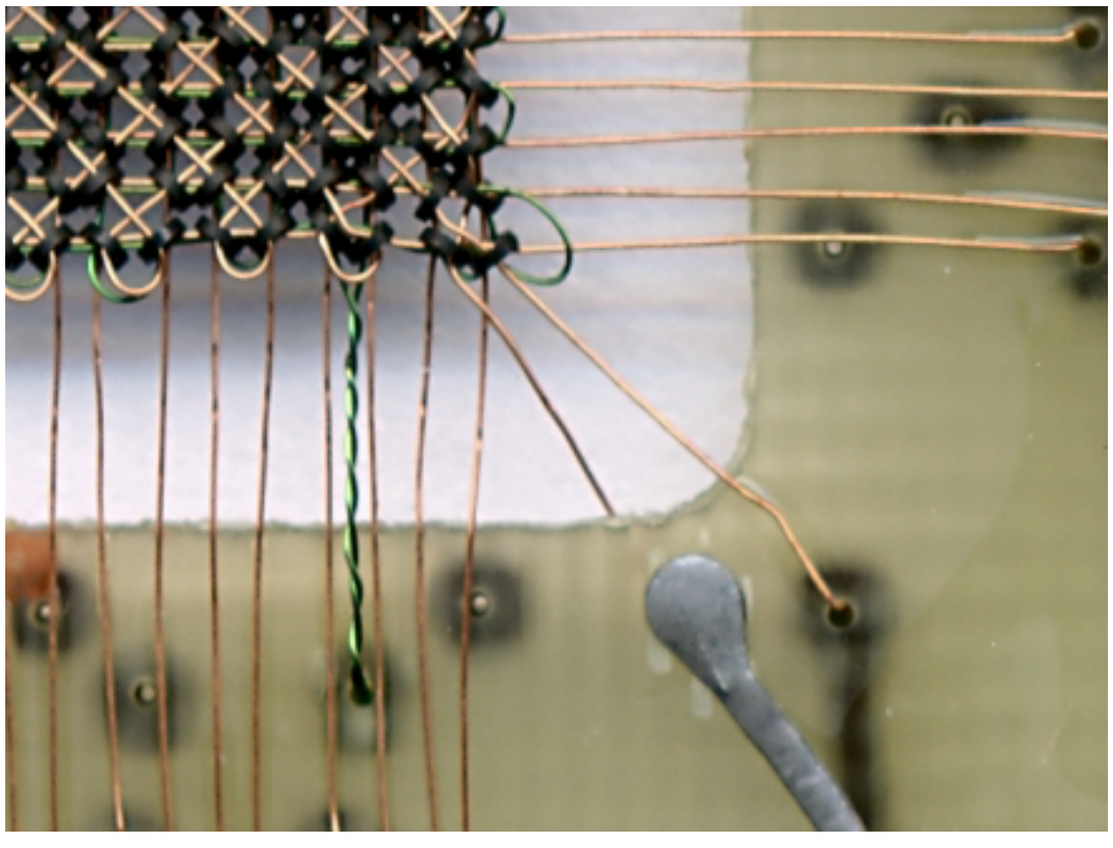
TIME 09:52:03
EPS1B PROCNT FIN 09:50:41
ROLBAK FIN 09:50:50
▶FS TAPES
FS KEVIN - TAPES DOES NOT EXIST. INPUT IGNORED
▶SO PRBIG
PRBIG: 7 FILES 675 PAGES
▶SO PRBIG *
PRBIG: UNUSLA SPERRY*TEMPER 86 EPGS D
PRBIG: EXEC 8 SYS$*TEMPER 94 EPGS D
PRBIG: EXEC 8 SYSAMB*ONSITEPRINTS 7 EPGS J
PRBIG: EXEC 8 SYS$*GP082UD01 158 EPGS M
PRBIG: BT104 SECURE*BT100UL03 5 EPGS M
PRBIG: EXEC 8 SYSAMA*GP082UD01 310 EPGS M
PRBIG: EXEC 8 SYS$*BT100UL03 15 EPGS M
▶T B
ACTIVE RUNS: 09:51:53 OPEN/MAXOPN=8/10
UNUNCB/+21 UNUTIM/+6 UNUCMS/+30
EP6503/+30 UNCLZR/+125 EP6011/+26
EP6F9/+26 XA710A/+26
▶SO PRT77 *
EP122/000092 START 09:51:58
AEZ8N/000850 START 09:52:03

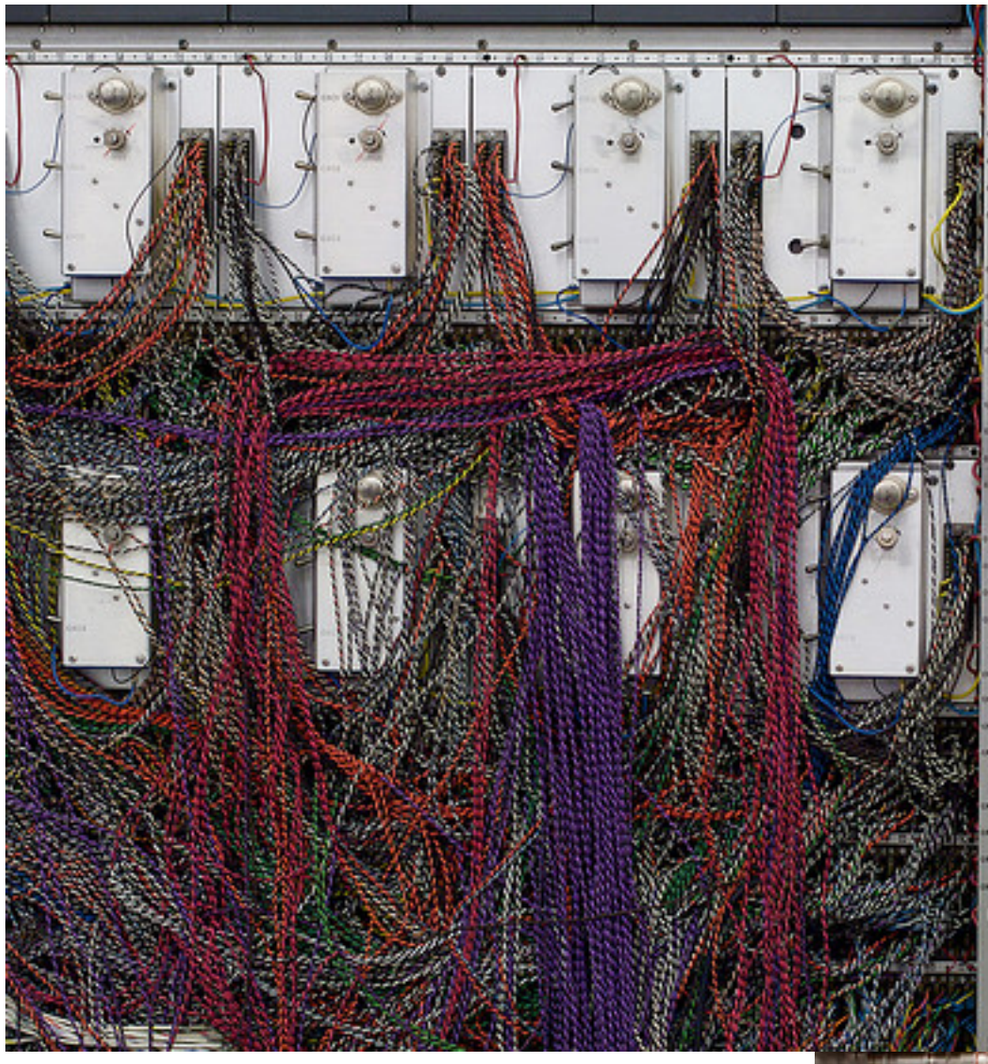




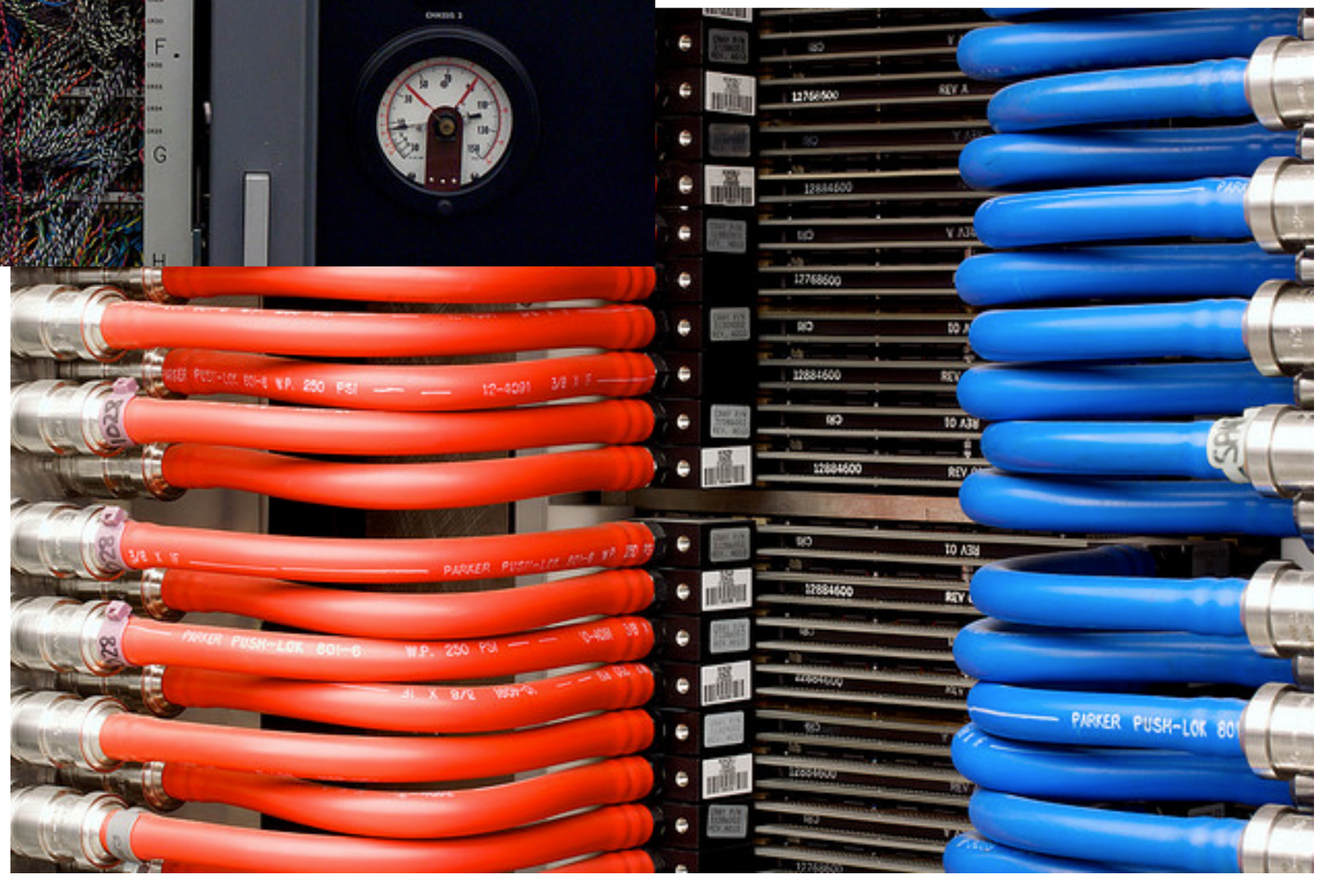


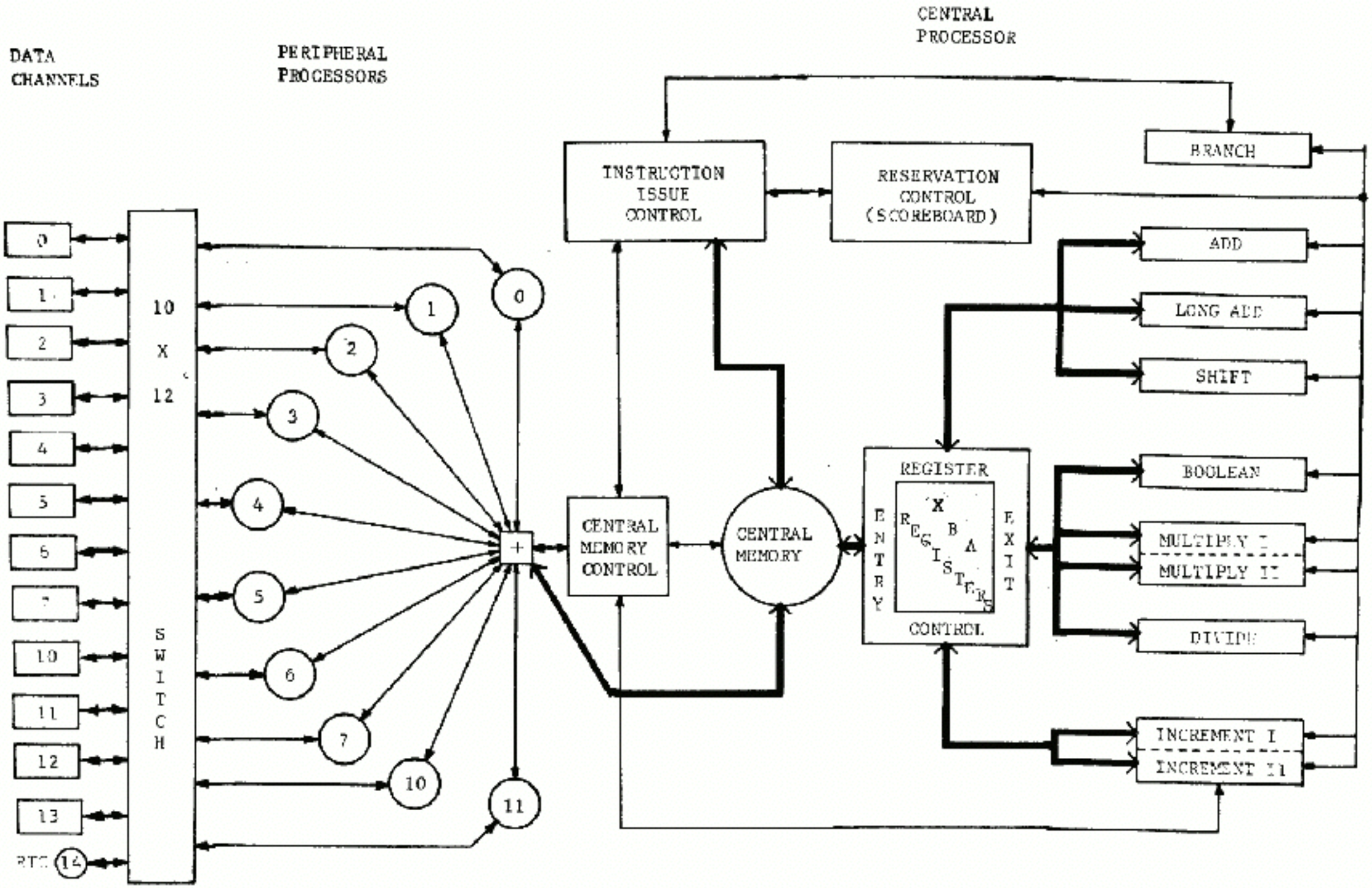

NSA-26798
0 1 cm 2 3 4 5 6 7 8 9 10 11






CONDENSER
CHASSIS 1
CHASSIS 2


PARKER PUSH-LOK 801-6 W.P. 250 PSI
PARKER PUSH-LOK 801-6 W.P. 250 PSI
PARKER PUSH-LOK


DATA
CHANNELS
PERIPHERAL
PROCESSORS
CENTRAL
PROCESSOR
0
1
2
3
4
5
6
7
10
11
12
13
RTC
14
10
X
12
SWITCH
0
1
2
3
4
5
6
7
10
11
+
INSTRUCTION
ISSUE
CONTROL
RESERVATION
CONTROL
(SCOREBOARD)
CENTRAL
MEMORY
CONTROL
CENTRAL
MEMORY
REGISTER
ENTRY
EXIT
CONTROL
BRANCH
ADD
LONG ADD
SHIFT
BOOLEAN
MULTIPLY I
MULTIPLY II
DIVIDE
INCREMENT I
INCREMENT II

Why bother with executing out of order?

- OOO: To execute instructions in a different order than the compiler (or person) has specified
- If much work is contending for a limited resource, it would be nice to make progress on other work that did *not* used that resource
  - e.g. bottlenecked on memory
- Instruction Level Parallelism, makes you go faster!
- Memory today has *variable* latency
- Code performance migratability

Mem->Issue

- WaitFor:
  - Instruction bits
    - Space to issue the instruction to
- Do
  - Issue it!

Issue->Dispatch

- WaitFor:
  - The functional unit must be free
  - The output register must be free
- Do:
  - Move the instruction to the functional unit

Dispatch->Execute

- WaitFor:
  - have the instruction has to be there
    - input registers have to be valid
- Do:
  - read the registers and execute!

Execute -> Complete

- WaitFor:
  - execution to complete
- Do:
  - post the write to the register file

Complete

- WaitFor
- Do

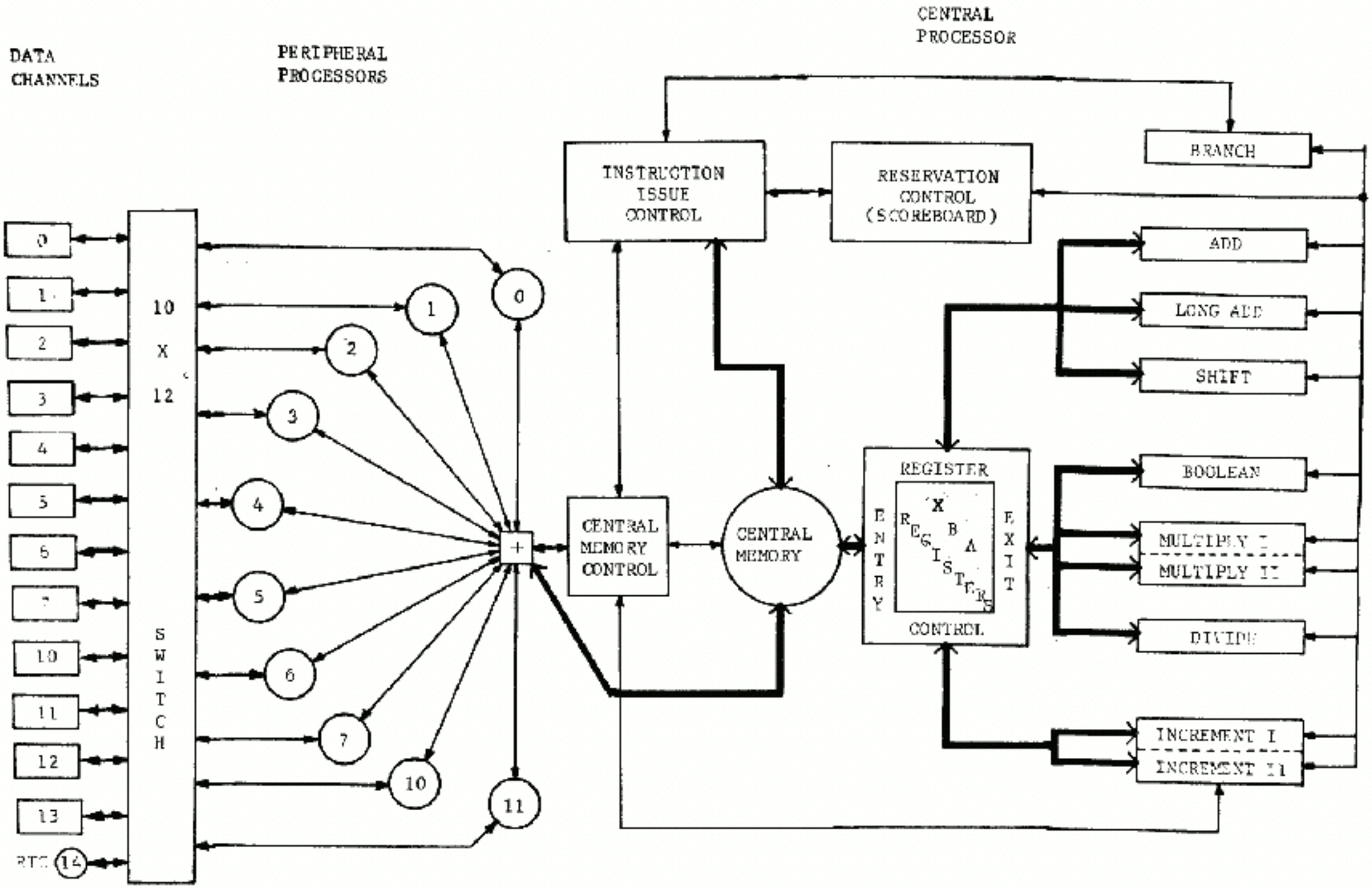
troubles

- DIV r1, r2 -> R3
  ADD r4,r5 -> r1
  STORE r1, @(r6)

- Four solutions:
  - Solution: don't do parallel stuff
  - Completion unit
  - *Very* carefully dispatch instructions so later instructions only get dispatched if they have a latency that is long enough not to modify architectural state before earlier instructions.
  - Accept it.

What are the “I/O” “processors”?

- Accelerators for I/O
  - Provided flexibility to I/O
    - DMA read/write
    - Asynchronous to the main CPU
    - Shared main memory with the CPU
- A little micro controller
  - FGMT, Fine-grained Multithreading


DATA
CHANNELS
PERIPHERAL
PROCESSORS
CENTRAL
PROCESSOR
0
1
2
3
4
5
6
7
10
11
12
13
RTC
14
10
X
12
SWITCH
0
1
2
3
4
5
6
7
10
11
+
INSTRUCTION
ISSUE
CONTROL
RESERVATION
CONTROL
(SCOREBOARD)
CENTRAL
MEMORY
CONTROL
CENTRAL
MEMORY
REGISTER
ENTRY
EXIT
CONTROL
BRANCH
ADD
LONG ADD
SHIFT
BOOLEAN
MULTIPLY I
MULTIPLY II
DIVIDE
INCREMENT I
INCREMENT II

Why FGMT?

- "Application level" threading
- The logic is precious, the register state, not so much => hence the overhead is small
- A way to deal with long latency operations
- Circuit world variation: C-slow retiming