CDC 6600
DEAD START PANEL

FIGURE 26  Cabinet top view.
TIME 09:52:03

FS TAPES
FS KEYIN - TAPES DOES NOT EXIST, INPUT IGNORED

PSQ PRBIG
PRBIG: 7 FILES 675 PAGES

PRBIG:
PRBIG: UMUSLA SPERRY=TEMPER 86 EPGS D
PRBIG: EXEC 8 SYS**=TEMPER 94 EPGS D
PRBIG: EXEC 8 SYS**=ONBITPRINTS 7 EPGS J
PRBIG: EXEC 8 SYS**=GP082UD01 158 EPGS N
PRBIG: EXEC 8 SYS**=BT100UL03 5 EPGS N
PRBIG: EXEC 8 SYS**=GP082UD01 310 EPGS N
PRBIG: EXEC 8 SYS**=BT100UL03 15 EPGS N

T B
ACTIVE RUNS: 09:51:53 OPEN/MAXOPEN=8/10
UNUCN/6
UNUCB/21
EP603/30
EP606/26

PSQ PR777 *
EP122/980092 START 09:51:59
AEZRX/980850 START 09:52:03
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 use 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
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