Chapter 4

Computer Organization

Von Neuman Concept

- Stored program concept
- General purpose computational device driven by internally stored program
- Data and instructions look same i.e. binary
- Operation being executed determined by HOW we look at the sequence of bits
  - Fetch
  - Decode View bits as instruction
  - Execute

\[\text{Data} \text{ might be fetched as a result of execution}\]
Von Neuman Architecture

- CPU
  - ALU
  - Control Unit
- I/O Buses
- Memory Unit
- Devices

Von Neuman Machine Architecture

CPU = ALU + Cntrl Unit

ALU
- Functional Unit
- Instruction set
- Arithmetic & Logic
- Registers
- Intermediate storage

Cntrl Unit
- fetch
- decode
- execute

Address Bus / Data Bus wires over which Instr / data is transferred from memory to ALU

Von Neumann Bottleneck
CPU: **ALU Component**

- Assumes instruction format: OP code, LHO, RHO
  - Instruction / data fetched & placed in register
  - Instruction / data retrieved by functional unit & executed
  - Results placed back in registers
- Control Unit sequences the operations

CPU: **Control Unit Component**

- Fetch Unit
  - Get instruction at location pointed to by PC and place in IR
- Decode Unit
  - Determine which instruction & signal hardware that implements it
- Execute Unit
  - Hardware for instruction execution (could cause more data fetches)
Fetch - Execute cycle

```plaintext
PC = <machine start address>
IR = memory[PC]
haltFlag = CLEAR
while (haltFlag not SET during execution) {
    execute(IR):
    PC = PC + 1;
    IR = memory[PC]
}
```

OS boot-up...

- How does the system boot up?
  - Bootstrap loader
  - OS
  - Application
A Bootstrap Loader

The power-up sequence

\[\text{load PC, FIXED_LOC}\]

Where FIXED_LOC addresses the bootstrap loader (in ROM).

The bootstrap loader has the form:

\[
\begin{align*}
\text{load R1, } & -0 \\
\text{load R2, } & = \text{LENGTH_OF_TARGET} \\
\text{loop: } & \text{read R1, FIXED_DISK_ADDRESS} \\
& \text{store R1, [FIXED_DEST, R1]} \\
& \text{incr R1} \\
& \text{bleq R1, R2, loop} \\
& \text{br FIXED_DEST}
\end{align*}
\]

Address of BS Loader

<table>
<thead>
<tr>
<th>Fetch</th>
<th>Decode</th>
<th>Execute</th>
<th>Branches to OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reads</td>
<td>OS in</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory Unit

![Diagram of Memory Unit](image)

Memory Unit
Memory Unit

- Memory Unit contains
  - Memory
    - Instructions & Data
  - MAR (Memory Address Register)
  - MDR (Memory Data register)
  - CMD (Command Register)
  - Get instruction at location pointed to by PC and place in IR

Memory Access

- Read from Memory
  - MAR ← MemAddr
  - CMD ← ‘Read OP’ (from IR)
  - Execute
    - MDR ← Mem[ MAR ]

- Write to Memory
  - MAR ← MemAddr
  - CMD ← ‘Write OP’ (from IR)
  - Execute
    - Mem[ MAR ] ← MDR
Device & Device Controller

- Central Processing Unit (CPU)
  - Arithmetic-Logical Unit (ALU)
  - Control Unit
- Address Bus
- Data Bus
- Primary Memory (ROM) (Executable Memory)
- Device Controller
- Device
- Interfaces
- Device Driver
- In OS

Device Controller-Software Relationship

- Application Software
- High-Level I/O Machine
- Device Controller
- Device
- Bus
- PCI
- SCSI
- Standard Interface
- Device driver
Device Controller Interface

- Driver places command if status “Done”
- Driver interrogated these to check status of device
- Interface to driver

Device Controller

- Device controller is a processor and allows 2 parts of the process to proceed concurrently

Program

Controller

write

Prints info
OS could provide higher level operations to application than the one Driver presents to it

Interface presented by Driver to Application program thru OS

Controller/Driver Interface

Controller/Device Interface

write(...)
How do interrupts factor in? ...

Scenario (3)
- Program: issues “write”
- Driver:
  - Suspend program until write is completed, then program is unsuspended

This is Interrupt-driven

Interrupts Driven Service Request
- Process is suspended only if driver/controller/device cannot service request
- If a process is suspended, then, when the suspended process’ service request can be honored
  - Device interrupts CPU
  - OS takes over
  - OS examines interrupts
  - OS un-suspends the process

- Interrupts
  - Eliminate busy wait
  - Minimizes idle time
Interrupts ...

Interrupt Handler in OS: disables interrupts
:          : Interrupt processed
:          : enables interrupts

What if multiple devices (or 2nd device) sends interrupt while the OS is handling prior interrupt?

If priority of 2nd interrupt higher than 1st then 1st interrupt suspended → 2nd interrupt handled → Resumption of handling 1st interrupt