Computer Organization

COMPUTER ORGANIZATION: AN OVERVIEW
COMPUTER MEMORY
COMPUTER PROCESSOR
IMPLEMENTING COMPUTERS
The Von Neumann Architecture: An Abstract View

Also known as the stored-program architecture
The Von Neumann Architecture: A More Detailed View

**Figure 5.2** Components of the Von Neumann Architecture
The Von Neumann Architecture: A Really Detailed View

Figure 5.18 The Organization of a Von Neumann Computer
The Von Neumann Architecture: An Operational View

The Von Neumann Execution Cycle:

repeat
  Fetch next instruction
  Decode instruction
  Execute instruction
Computer Memory: Overview

- Focus here on (volatile) Random-Access Memory (RAM), cf. (non-volatile) Read-Only Memory (ROM).

- Three characteristics of RAM:
  1. Divided into fixed-width cells, each of which has a unique unsigned-integer address $0, 1, 2, \ldots, \text{MAX}$ (address space).
  2. The cell is the minimal unit of fetch / store access.
  3. All cells have the same access time.

- Crucial to distinguish a memory address and the contents of memory at a particular address, e.g.,

  $\text{address} \implies 5743_{10}: -29_{10} \leftrightarrow \text{contents}$
Computer Memory: Overview (Cont’d)

- Standard cell-width $W = 8$ bits (byte); standard address = 32 or 64 bits; standard access time $\approx 5$-10 nanoseconds.
- Memory size stated in terms of number of bytes:

  Kilobyte (KB) $= 10^3$ (thousand) bytes
  Megabyte (MB) $= 10^6$ (million) bytes
  Gigabyte (GB) $= 10^9$ (billion) bytes
  Terabyte (TB) $= 10^{12}$ (trillion) bytes
  Petabyte (PB) $= 10^{15}$ (quadrillion) bytes
  Exabyte (EB) $= 10^{18}$ (quintillion) bytes

...
Computer Memory: Overview (Cont’d)

- All communication done via the **Memory Address Register (MAR)** and the **Memory Data Register (MDR)**.

- Two basic operations:
  1. Fetch(address):
     1. Load address into MAR
     2. Decode address in MAR
     3. Copy cell contents at address into MDR
  2. Store(address, value):
     1. Load address into MAR
     2. Load value into MDR
     3. Decode address in MAR
     4. Copy MDR value into addressed cell
Computer Memory: Internal Structure (1D Abstract)

Figure 5.3
Structure of Random Access Memory
Computer Memory: Internal Structure (1D)

Figure 5.5 Organization of Memory and the Decoding Logic
Computer Memory: Internal Structure (2D Abstract)

Figure 5.6 Two-Dimensional Memory Address Organization
Computer Memory: Internal Structure (2D)

Figure 5.7
Overall RAM Organization
Computer Memory: The Memory Hierarchy

Memory speed lags behind CPU speed
Computer Memory:
The Memory Hierarchy (Cont’d)

Deal with processor / memory speed differences as follows:

1. **Registers**: Communicate with(in) processor; contain currently-executed instruction and data and associated information.

2. **Cache**: Communicates with registers and primary; uses principle of locality to pre-load anticipated instructions and data from primary.

3. **Primary**: Communicates with cache and secondary; contains programs being executed and their data.

4. **Secondary**: Communicates with primary; contains all programs and data of interest.

I/O interface devices, e.g., keyboards, screens, are treated as secondary memory devices.
Computer Memory: The Memory Hierarchy (Cont’d)
Computer Memory: The Memory Hierarchy (Cont’d)

- Deal with very large access-time difference between primary and secondary memory using an I/O controller, a special-purpose computer consisting of one or more I/O buffers and associated control logic.
- When fetching from secondary, the I/O controller loads data from the appropriate device into the buffer and, when full, sends the buffer’s contents to the processor.
- When storing to secondary, the I/O controller loads data from the processor into the buffer and sends the buffer’s contents to the appropriate device.
- Special interrupt signals used to let the processor know when I/O operations are done.
Computer Memory: The Memory Hierarchy (Cont’d)

Figure 5.9
Organization of an I/O Controller
Computer Processor: Overview

- Two main parts:
  1. **Arithmetic Logic Unit (ALU):** Performs arithmetic and logical operations.
  2. **Control Unit:** Handles interpretation and execution of program instructions. This involves directing the operations of the ALU and memory as well as interacting with the I/O controller.

- Both the ALU and the Control Unit have their own associated groups of special-purpose registers associated with their internal operations.
Computer Processor: The Arithmetic Logic Unit (ALU)

- Two types of ALU registers:
  1. **Value Registers (R0, R1, R2, . . .)**: A set of 16–128 registers which contain data for current and upcoming operations as well as intermediate results.
  2. **Condition Code Register (CCR)**: A collection of bits specifying the results (1 if true, 0 if false) of the most recently executed value comparison, e.g., LT (less-than), EQ (equal-to), GT (greater-than).

- Value registers communicate with memory and the ALU and can specified as either the left or right operand.
- The CCR communicates with the ALU, which passes the value of any condition-bit as requested to the control unit.
Computer Processor:
The Arithmetic Logic Unit (ALU) (Cont’d)

Figure 5.11 Multiregister ALU Organization
Computer Processor: 
The Arithmetic Logic Unit (ALU) (Cont’d)

Figure 5.13 Overall ALU Organization
Computer Processor: The Control Unit

- Two Control Unit registers:
  1. **Program Counter (PC)**: Holds address in memory of next instruction to be executed.
  2. **Instruction Register (IR)**: Holds the current instruction being executed. This includes not only the op-code ($IR_{op}$) but the addresses of the instruction operands ($IR_{add}$, e.g., memory / ALU value registers).

- Instruction decoder circuitry uses the the $k$-bit opcode in the instruction in the IR to specify the appropriate one of the $2^k$ signals to that instruction’s execution circuitry and/or other computer components.
Computer Processor: The Control Unit (Cont’d)

Figure 5.16
Organization of the Control Unit Registers and Circuits
Computer Processor: Machine Language

- An instruction = op-code + 0–3 address fields, e.g.,

<table>
<thead>
<tr>
<th>op-code</th>
<th>address-1</th>
<th>address-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>000101</td>
<td>000000110011</td>
<td>000010001100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COMPARE</th>
<th>Addr1</th>
<th>Addr2</th>
</tr>
</thead>
</table>

- Is part of either Reduced or Complex Instruction Set Computer (RISC / CISC) machine language; differ in tradeoff of required hardware vs. resulting program size.
Four types of machine language instructions:

1. **Data Transfer**: Move values between memory cells and/or ALU registers, e.g., \textsc{LOAD Addr1}, \textsc{LOAD Addr2}, \textsc{MOVE Addr1 Addr2}.

2. **Arithmetic**: Perform arithmetic / logical operations on values in memory cells and/or ALU registers, e.g., \textsc{ADD Addr1 Addr2 Addr3}, \textsc{ADD Addr1 Addr2}.

3. **Comparison**: Compare two values and set CCR bits, e.g., \textsc{COMPARE Addr1 Addr2}.

4. **Branch**: Alter next instruction to be executed (often on basis of preceding comparison), e.g., \textsc{JUMP Addr1}, \textsc{JUMPGT Addr1}, \textsc{HALT}.
Computer Processor:
Machine Language (Cont’d)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>value of a</td>
</tr>
<tr>
<td>101</td>
<td>value of b</td>
</tr>
<tr>
<td>102</td>
<td>value of c</td>
</tr>
</tbody>
</table>

- set \( a \) to value of \( b + c \)

<table>
<thead>
<tr>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
</tr>
<tr>
<td>51</td>
</tr>
<tr>
<td>52</td>
</tr>
</tbody>
</table>

if \( a > b \) then

- set \( c \) to value of \( a \)

else

- set \( c \) to value of \( b \)

<table>
<thead>
<tr>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
</tr>
<tr>
<td>61</td>
</tr>
<tr>
<td>62</td>
</tr>
<tr>
<td>63</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>65</td>
</tr>
</tbody>
</table>
The Von Neumann Architecture: A Detailed View Redux

Figure 5.18 The Organization of a Von Neumann Computer
The Von Neumann Architecture: An Operational View Redux

The Von Neumann Execution Cycle:

while no \texttt{HALT} or fatal error do
  Fetch next instruction
  Decode instruction
  Execute instruction

- Let us consider the details of this cycle in the context of a 16-instruction RISC machine language for a computer with a single-register ALU.
## OC Instruction Meaning

<table>
<thead>
<tr>
<th>OC</th>
<th>Instruction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LOAD Addr</td>
<td>( CON(Addr) \rightarrow R )</td>
</tr>
<tr>
<td>1</td>
<td>STORE Addr</td>
<td>( R \rightarrow CON(Addr) )</td>
</tr>
<tr>
<td>2</td>
<td>CLEAR Addr</td>
<td>( 0 \rightarrow CON(Addr) )</td>
</tr>
<tr>
<td>3</td>
<td>ADD Addr</td>
<td>( R + CON(Addr) \rightarrow R )</td>
</tr>
<tr>
<td>4</td>
<td>INCREMENT Addr</td>
<td>( CON(Addr) + 1 \rightarrow CON(Addr) )</td>
</tr>
<tr>
<td>5</td>
<td>SUBTRACT Addr</td>
<td>( R - CON(Addr) \rightarrow R )</td>
</tr>
<tr>
<td>6</td>
<td>DECREMENT Addr</td>
<td>( CON(Addr) - 1 \rightarrow CON(Addr) )</td>
</tr>
<tr>
<td>7</td>
<td>COMPARE Addr</td>
<td>if ( CON(Addr) &gt; R ) then ( GT = 1 ) else 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if ( CON(Addr) = R ) then ( EQ = 1 ) else 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if ( CON(Addr) &lt; R ) then ( LT = 1 ) else 0</td>
</tr>
</tbody>
</table>
The Von Neumann Architecture:
An Operational View Redux (Cont’d)

<table>
<thead>
<tr>
<th>OC</th>
<th>Instruction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>JUMP Addr</td>
<td>Addr $\rightarrow$ PC</td>
</tr>
<tr>
<td>9</td>
<td>JUMPGT Addr</td>
<td>if $GT = 1$ then Addr $\rightarrow$ PC</td>
</tr>
<tr>
<td>10</td>
<td>JUMPEQ Addr</td>
<td>if $EQ = 1$ then Addr $\rightarrow$ PC</td>
</tr>
<tr>
<td>11</td>
<td>JUMPLT Addr</td>
<td>if $LT = 1$ then Addr $\rightarrow$ PC</td>
</tr>
<tr>
<td>12</td>
<td>JUMPNEQ Addr</td>
<td>if $EQ = 0$ then Addr $\rightarrow$ PC</td>
</tr>
<tr>
<td>13</td>
<td>IN Addr</td>
<td>Store input value at Addr</td>
</tr>
<tr>
<td>14</td>
<td>OUT Addr</td>
<td>Output $CON(\text{Addr})$</td>
</tr>
<tr>
<td>15</td>
<td>HALT</td>
<td>Stop program execution</td>
</tr>
</tbody>
</table>

See page 261 of textbook for notations in “Meaning” column.
The Von Neumann Architecture: An Operational View Redux (Cont’d)

set \( a \) to value of \( b + c \)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>LOAD 101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>ADD 102</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>STORE 100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\begin{align*}
100 & \left\{ \text{value of } a \right. \\
101 & \left\{ \text{value of } b \right. \\
102 & \left\{ \text{value of } c \right. \\
\cdots & \end{align*}
The Von Neumann Architecture: An Operational View Redux (Cont’d)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>value of (a)</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>value of (b)</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>value of (c)</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{if } a > b \text{ then} & \quad 60 \quad \text{LOAD 101} \\
& \quad \text{COMPARE 100} \\
\text{set } c \text{ to value of } a & \quad 61 \quad \text{JUMPGT 66} \\
\text{else} & \quad 62 \quad \text{LOAD 101} \\
\text{set } c \text{ to value of } b & \quad 63 \quad \text{STORE 102} \\
& \quad 64 \quad \text{LOAD 101} \\
& \quad 65 \quad \text{JUMP 68} \\
& \quad 66 \quad \text{LOAD 100} \\
& \quad 67 \quad \text{STORE 102} \\
& \quad 68 \quad \cdots
\end{align*}
\]
The Von Neumann Architecture: An Operational View Redux (Cont’d)

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>value of $IND$</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>value of $MAXIND$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>set $IND$ to 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>while $IND \leq MAXIND$ do</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>set $IND$ to $IND + 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td><code>CLEAR 100</code></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td><code>LOAD 101</code></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td><code>COMPARE 100</code></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td><code>JUMPGT 77</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td><code>INCREMENT 100</code></td>
<td></td>
</tr>
<tr>
<td>76</td>
<td><code>JUMP 61</code></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
The Von Neumann Architecture: An Operational View Redux

Details of the three phases of the Von Neumann cycle:

1. **Fetch next instruction**: Load next instruction into IR and update PC, i.e.,
   
   1. \( PC \rightarrow MAR \)
   2. \( FETCH \)
   3. \( MDR \rightarrow IR \)
   4. \( PC + 1 \rightarrow PC \)

2. **Decode instruction**: Determine which circuitry must be activated to execute the instruction, i.e.,
   
   1. \( IR_{op} \rightarrow \text{instruction decoder} \)
The Von Neumann Architecture: An Operational View Redux

3. **Execute instruction**: Trigger the unique circuitry required to execute the instruction, e.g.,

   **LOAD** Addr
   1. \( IR_{addr} \rightarrow MAR \)
   2. FETCH
   3. \( MDR \rightarrow R \)

   **STORE** Addr
   1. \( IR_{addr} \rightarrow MAR \)
   2. \( R \rightarrow MDR \)
   3. STORE

   **ADD** Addr
   1. \( IR_{addr} \rightarrow MAR \)
   2. FETCH
   3. \( MDR \rightarrow ALU \)
   4. \( R \rightarrow ALU \)
   5. ADD
   6. \( ALU \rightarrow R \)
SSEM ("Baby")  
(1948, U. Manchester)  

EDSAC  
(1949, U. Cambridge)

SSEM and EDSAC were world’s first operational electronic stored-program computers.
Implementing Computers: Mainframes

IBM System/360 (1967)
Implementing Computers: Minicomputers

PDP I (1960)  
PDP 8 (1965)
Implementing Computers: Memory

Magnetic tape (1951)

Magnetic disk (1956)

Magnetic core (1953)

Massive cheap transistor-based storage possible in 1990s.
Implementing Computers: I/O Interfaces

Punch card / tape (1940s)
Teletype (1940s)
CRT Display (1940s)
Implementing Computers: Microprocessors

Instead of being a little mainframe, the PC is, in fact, more like an incredibly big chip. – Robert X. Cringely

The microprocessor was invented by Ted Hoff in 1971.
Implementing Computers: Microcomputers

Implementing Computers: Microcomputers (Cont’d)

- First true “laptop” PCs (GRiD Compass 1101 (1982) [$8K]; see above left) appear in 1980s, cf., portable “desktop” PCs like the Osborne I; expense of display and memory technologies limits market severely.
Implementing Computers: Microcomputers (Cont’d)

• Hand-held personal computing appears first as Personal Digital Assistants (PDAs) in early 1990s.

• Early PDAs (Palm Pilot, Newton) were typically too expensive and based on technologies of limited user interest, e.g., handwriting recognition.

• Second-generation PDAs achieve success among business and government users when combined with basic secure messaging abilities, e.g., Blackberry (1999).
The convergence of hand-held multimedia-enabled computing and communication technology has resulted in tablet computers and smartphones; the former is preferable for screen size and the latter for device size.
Implementing Computers: Non-Von Neumann Architectures

CM-2 (1987)  
DWave 2000Q (2017)

Based on massively parallel instruction execution by multiple processors (CM-2) or quantum entanglement (DWave 2000Q).
...And If You Liked This...

- MUN Computer Science courses on this area:
  - COMP 2003: Computer Architecture
  - COMP 4723: Introduction to Microprocessors
- MUN Computer Science professors teaching courses / doing research in this area:
  - Rod Byrne