The Von Neumann Model

Instructing the Computer

To perform a task, the computer is provided with program

Program
- Consists of a set of instructions that the computer must do complete the task

Instruction
- Smallest piece of work specified (fundamental unit)
  - One high-level statement in Java/C can be more than one instruction for the machine
  - Encoded as 0s and 1s
  - Either carried completely or not at all
  - This is because future instructions can be dependent on the previous

Humans vs. Computers

Scenario: Listening to a story and then tell it again.

Humans
- Listen
  - Input: ears
- Process
  - Cerebrum
- Put it in memory
  - Short or long term memory
- Repeat
  - Output: voice, hand gestures
- Controller
  - Brain

Computer
- Listen
  - Input: Keyboard, Microphone
- Process
  - Central Processing Unit (CPU)
- Put in memory
  - RAM or Hard Disk
- Repeat
  - Output: Monitor, Speaker
- Controller
  - Control Unit

Von Neumann Model (stored program concept)

- Components to process a program:
  - Memory: where instructions and data are stored
  - Control unit:
    - Co-ordinates all other units
    - It also interprets instructions
  - Processing unit: for performing arithmetic and logical operations
  - Input/Output units: for interacting with the real world
Memory

$2^n \times m$ array of stored bits

**Address**
- unique ($k$-bit) identifier of location

**Contents**
- $m$-bit value stored in location

**Basic Operations:**
- **LOAD**
  - read a value from a memory location
- **STORE**
  - write a value to a memory location

Processing Unit

**Functional Units**
- ALU = Arithmetic and Logic Unit
- Could have many functional units (some special-purpose, e.g., multiply, square root, ...)
- LC-3: ADD, AND, NOT

**Registers**
- Small, temporary storage
- Operands (data to be operated on) and results of functional units
  - LC-3: eight user register (R0, ..., R7)
- Some more for book keeping and interfacing

**Word Size**
- Number of bits normally processed by ALU in one instruction
- Also width of registers
- LC-3: 16 bits

Interface to Memory

How does processing unit get data to/from memory?

**MAR:** Memory Address Register
**MDR:** Memory Data Register

To read a location A
1. Write the address A into the MAR
2. Send a “read (r)” signal to the memory
3. Read the data from MDR

To write a value X to a location A
1. Write the data X to the MDR
2. Write the address A into the MAR
3. Send a “write (w)” signal to the memory

So what is the need for the interface?

Control Unit

Orchestrates execution of the program (like your Brain)
- Keeps track of where we are in the process of executing
  - The program as well as each instruction

**Instruction Register (IR)**
- Contains the current instruction

**Program Counter (PC)**
- Contains the address of the next instruction to execute

**Control Unit**
- Reads an instruction from memory (at PC)
- Interprets the instruction
- Generates signals that tell the other components what to do
- Instruction may take many machine cycles to complete
Instructions

Instruction: **Fundamental unit of work**

**Constituents**
- **Opcode**: operation to be performed (e.g., +, & )
- **Operands**: data/locations to be used for operation
  - Source: location that contains the data
  - Destination: location that will store the result of computation
  - Immediate: data values not contained at a particular location

Encoded as a sequence of bits (**just like data!**)
- Sometimes have a fixed length (e.g., 16 or 32 bits)
- Control unit interprets instruction
  - Generates control signals to carry out operation
  - Atomic: operation is either executed completely, or not at all

**Example**

**LC-3 has 16-bit instructions**
- Each instruction has a four-bit opcode, bits [15:12]
- Has eight registers (R0-R7) for temporary storage

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Dst</td>
<td>Src1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Src2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LC-3 ADD**: Add the contents of R2 to the contents of R6, and store the result in R6.

Instruction Set Architecture (ISA)

**ISA = Programmer-visible components & operations**
- Memory organization
  - Address space - # of locations?
  - Addressability - how many bits per location?
- Register set
  - How many? What size? How are they used?
- Instruction set
  - Opcodes: Which operation?
  - Data types: int, floating point etc
  - Addressing modes: how to calculate the effective memory address of an operand?

All information needed to write/generate machine language program
- You care if you write e.g. compilers for high-level language

Instruction Processing

**How are instructions executed?**
- Each instruction goes to 6 stages
- Some instructions may not use every stage

1. FETCH instruction from mem.
2. DECODE instruction
3. EVALUATE ADDRESS
4. FETCH OPERANDS
5. EXECUTE operation
6. STORE result

**Example**

**LC-3 ADD**: Add the contents of R2 to the contents of R6, and store the result in R6.
Instruction Processing: FETCH

Idea
- Put next instruction in IR & increment PC

Steps
- Load contents of PC into MAR
- Increment PC (by default to the next instruction)
- Send “read” signal to memory
- Read contents of MDR, store in IR

Who makes all this happen?
- Control unit

Instruction Processing: DECODE

Identify opcode
- Using the opcode bits
  - In LC-3, always first four bits of instruction

Identify operands from the remaining bits

Control unit implements DECODE

Instruction Processing: EVALUATE ADDRESS

Compute memory address
- For loads (reading to mem) and stores (writing to mem)
- Addressing is not straightforward
  - E.g. Reference memory address is known (base)
    - If you want to go to the 4th location from base then offset is set to 4

Example: LC-3 LDR Instruction

Reads data from memory
Base + offset addressing mode
- Add offset to base register to produce memory address
- Load from memory address into destination register

“Add the value 6 to the contents of R3 to form a memory address. Load the contents of memory at that address and place the resulting data in R2.”
Instruction Processing: OPERAND fetch

Get source operands for operation

Examples
- Read data from register file (e.g., for an ADD)

Instruction Processing: EXECUTE

Actually perform operation
- ALU receives the operands
- E.g., ADD/SUB, AND, OR, NOT, MULT
- Do nothing (e.g., for loads(reads) and stores(writes))

Instruction Processing: STORE

Write results to destination
- Register or memory

Example
- Result of ADD is placed in destination reg.
- If we have limited space
  - Some values from registers are written back to memory via STORE command

Note: The store command will set MDR and assert WRITE signal to memory

Changing the Sequence of Instructions

Increment of PC
- FETCH phase always increments PC to the next instruction (PC + 1 in LC3)

We can also skip instructions:
- Programming constructs that change the sequence of instruction execution like If-then, loops etc.
- Instructions that change PC
  - Branches are conditional
    - Change the PC only if some condition is true e.g., the contents of a register is zero
  - Jumps are unconditional -> Always change the PC
Stopping the Computer

How does the computer know its done after x number instructions?
- e.g. A*B+C – 3 loads + 1 MULT + 1ADD = 5 total instructions

Modern machines
- User programs execute under the control of O.S.
  - Once the program terminates, the control instruction changes PC to again start O.S.
  - ISAs have HALT instruction to give back control

Instruction Processing Summary

Instructions look just like data
- Interpreted by machine

Three basic kinds of instructions
- Computational instructions (ADD, AND, …)
- Data movement instructions (LD(load), ST(store), …)
- Control instructions (JMP, BRnz, …)

Six basic phases of instruction processing
F → D → EA → OP → EX → S
- Not all phases are needed by every instruction