An Introduction to System Software and Virtual Machines

Chapter 6.1-6.3

Topics:
System Software
Assemblers and Assembly Language

The Naked Machine

• Difficult to use:
  ✓ Store program in RAM
  ✓ Put address of first instruction in PC, ...

• Difficult to program:
  • Machine language instructions look like: 1010000 ...
User Interfaces

- User interfaces
  - Hide the details of hardware (users require no in-depth knowledge of hardware), thus, allow easy access to the hardware resources.
- Use all the time in our daily life, e.g.:
  - Dashboard in a car
  - Control of a stereo/VCR
  - Punch keys on a microwave

System Software

- System software provides us with a simpler interface to operate and program the computer:
  - Is a collection of programs that manage the resources of the computer, and act as an intermediary between the user and the computer.
  - Hide the details of the Von Neumann architecture
  - Present information in understandable way
  - Allow user to access the hardware resources in a simple, safe, and efficient way.

Virtual Machine

- The services (interface) provided by the system software is what the user sees, that environment is called, a virtual machine (or virtual environment).
Typical System Software

- Language translators
  - Assemblers, compilers.
- Memory managers
  - Allocate space and load programs into memory.
- File systems
  - Storage/Retrieval of information from mass-storage devices
- Scheduler
  - Schedules the order of execution of programs.
- Utilities
  - E.g. text editors.

Using the Machine

- We want to write and run a program:
  - Use a text editor to create the program.
  - Store the file on the file system.
  - Use a language translator (compiler) to translate program into machine code.
  - Memory manager, or loader, allocates space and loads program into memory (RAM).
  - Scheduler, executes the program.
- We are interacting with the system software!

Programming the Machine

- Algorithms/Programs must be translated into machine-code before they can run on the computer:

  Pseudo-code \[ T_1 \] Programming Language

  T1: by a programmer
  T2: by a computer program

  Machine Code
Programming the Machine

- Instead of writing in machine code (yuck!) we can write our programs using a more "friendly" programming language:
  - Assembly language (learn now)
  - C++ (learn later)
- System software provides us with software tools to translate programs into machine code:
  - Assembler
  - Compiler

Assembly Language

- Similar instruction as in machine-code, except:
  - Can use symbolic names for instructions, addresses
  - Values can be stated as decimal
  - Can use comments
- Much simpler to use, for example, instead of
  
  \texttt{0001 000001001001}

  we can write

  \texttt{LOAD A} -- Load value of variable A into register

Assembly Instruction Format

<table>
<thead>
<tr>
<th>Label:</th>
<th>Op-code mnemonic</th>
<th>Address field</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- Labels are used to mark the location of:
  - Instruction we need to JUMP to.
  - Memory locations (variables) we want to refer to.
- Op-code mnemonics
  - The instructions in the computer instruction set.
- Address field
  - The address the instruction works with, or more typically, a label indicating the address.
Instruction Set for Our Von Neumann Machine

<table>
<thead>
<tr>
<th>Opcode Mnemonic</th>
<th>Address</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD</td>
<td>X</td>
<td>CON(X) → R</td>
</tr>
<tr>
<td>STORE</td>
<td>X</td>
<td>R → CON(X)</td>
</tr>
<tr>
<td>CLEAR</td>
<td>X</td>
<td>0 → CON(X)</td>
</tr>
<tr>
<td>ADD</td>
<td>X</td>
<td>R + CON(X) → R</td>
</tr>
<tr>
<td>INCREMENT</td>
<td>X</td>
<td>CON(X) + 1 → CON(X)</td>
</tr>
<tr>
<td>SUBTRACT</td>
<td>X</td>
<td>R - CON(X) → R</td>
</tr>
<tr>
<td>INCREMENT</td>
<td>X</td>
<td>CON(X) + 1 → CON(X)</td>
</tr>
<tr>
<td>COMPARE</td>
<td>X</td>
<td>If CON(X) &gt; R then GT = 1 else 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If CON(X) = R then EQ = 1 else 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If CON(X) &lt; R then LT = 1 else 0</td>
</tr>
<tr>
<td>JUMP</td>
<td>X</td>
<td>Get next instruction from memory location X</td>
</tr>
<tr>
<td>JUMPGT</td>
<td>X</td>
<td>Get next instruction from memory loc. X if GT = 1</td>
</tr>
<tr>
<td>JUMPPX</td>
<td>X</td>
<td>xx = LT / EQ / NEQ</td>
</tr>
<tr>
<td>IN</td>
<td>X</td>
<td>Input an integer value and store in X</td>
</tr>
<tr>
<td>OUT</td>
<td>X</td>
<td>Output, in decimal notation, content of memory loc. X</td>
</tr>
<tr>
<td>HALT</td>
<td></td>
<td>Stop program execution</td>
</tr>
</tbody>
</table>

Additional Format

- In addition to the aforementioned instructions, we use three pseudo instructions (do not generate any machine-code):
  - `.BEGIN` indicates beginning of program
  - `.END` indicates end of program
  - `.DATA` reserves memory for a data value
- Can include comments, by using `--`
  - `LOAD A -- This is a comment!`

Typical Assembly Program Structure

```
BEGIN        -- Beginning of program
...
-- Machine instructions
Label:
...
HALT         -- Stop program
A: .DATA     -- Data declaration
...
... .DATA    -- Data declaration
...          
.END         -- End of program
```

Chapter 6: An Introduction to System Software ...
Practice Question #1

- Write an assembly program that reads in 2 numbers, adds them together, and outputs their sum (algorithm given below):

```
BEGIN
IN A -- Get values for A and B
IN B
LOAD A -- Set the value of C to (A+B)
ADD B
STORE C
OUT C -- Print the value of C
HALT -- Stop
A: DATA 0 -- Reserving memory for variables
B: DATA 0 -- A, B, and C.
C: DATA 0
END
```

Practice Question #2

- Write an assembly program that reads in 5 numbers and prints out their sum (algorithm given below):

```
BEGIN
SET the value of Sum to 0
SET the value of i to 1
WHILE i <= 5 DO
    GET a value for N
    SET the value of Sum to (Sum + N)
    ADD 1 to i
END OF LOOP
PRINT the value of Sum
STOP
```
### Practice Question #3

Write an assembly program that reads in 2 numbers, and prints out the larger of the two (algorithm given below):

1. Get values for A and B
2. If $A \geq B$ then
   - Print the value of $A$
   - Else:
   - Print the value of $B$
3. Stop

```assembly
BEGIN
  IN A -- Get values for A and B
  IN B
  LOAD B
  COMPARE A -- If A \geq B then
  JUMPLT Else
  OUT A -- Print the value of A
  JUMP Endif
Else:
  OUT B
Endif: HALT
A: DATA 0 -- Reserve memory for variables.
B: DATA 0
END
```
Translation

- An assembler translates assembly programs into machine code.
  - Converts symbolic op-codes to binary.
    - Simply a table-lookup.
  - Converts symbolic addresses to binary. Two passes:
    1. Establishing bindings between labels and addresses
    2. Convert references to labels to binary according to bindings.
- The resulting file with the machine code is called an object file.

Translation, Build Bindings

<table>
<thead>
<tr>
<th>Program</th>
<th>Location Counter</th>
<th>Bindings Labels</th>
<th>addr's</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop:</td>
<td>IN</td>
<td>Loop</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>LOAD</td>
<td>Done</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>COMPARE</td>
<td>X</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>JUMPLT</td>
<td>Done</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>JUMP</td>
<td>Loop</td>
<td>4</td>
</tr>
<tr>
<td>Done:</td>
<td>OUT</td>
<td>Y</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>HALT</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>X: .DATA</td>
<td>END</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LOADING

By a program called loader which
- reads instructions of an object program into RAM
- places the address of first instruction to Program Counter (PC) to initiate execution.