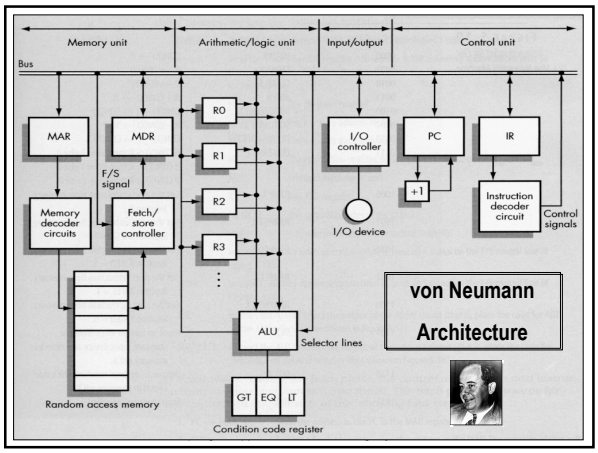
An Introduction to System Software and Virtual Machines

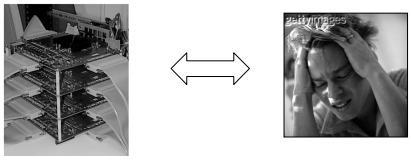
Chapter 6.1-6.3

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

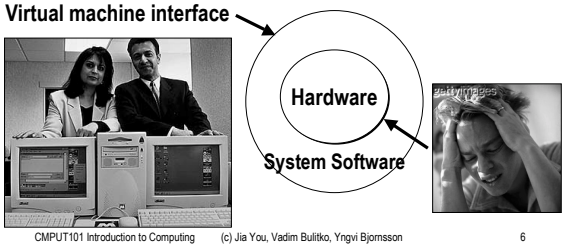
- User interfaces
  - Hide the details of hardware (users require no in-depth knowledge of hardware), thus, allow easy access to the hardware resources.
- Use all the time in our daily life, e.g.:
  - Dashboard in a car
  - Control of a stereo/VCR
  - Punch keys on a microwave

System Software

- System software provides us with an 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

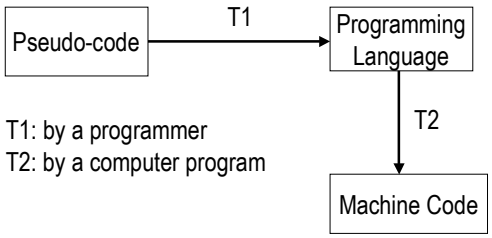
- Language translators
  - Assemblers, compilers.
- Memory managers
  - Allocate space and load programs into memory.
- File systems
  - Storage/Retrieval of information from mass-storage devices
- Scheduler
  - Schedules the order of execution of programs.
- Utilities
  - E.g. text editors.

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:



T1: by a programmer
T2: by a computer program

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

  0001 000001001001

  we can write

  LOAD A -- Load value of variable A into register

Assembly Instruction Format

Label:	Op-code mnemonic	Address field

- 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

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

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		
	.END		-- End of program

Practice Question #1

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

Get values for A and B
Set the value of C to (A+B)
Print the value of C
Stop

	.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):

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

	.BEGIN		
	CLEAR	Sum	-- Set the value of Sum to 0
	LOAD	One	-- Set the value of i to 1
	STORE	i	
Loop:	LOAD	Five	-- While i <= 5 do
	COMPARE	i	
	JUMPGT	Endloop	
	IN	N	-- Get the value of N
	LOAD	Sum	-- Set Sum to (Sum + N)
	ADD	N	
	STORE	Sum	
	INCREMENT	i	-- Add 1 to i
	JUMP	Loop	-- End of loop
Endloop:	OUT	Sum	-- Print the value of Sum
	HALT		-- Stop
Sum:	.DATA	0	-- Reserve memory for variables.
i:	.DATA	0	
N:	.DATA	0	
One:	.DATA	1	-- Constant 1
Five:	.DATA	5	-- Constant 5
	.END		

Practice Question #3

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

Get values for A and B
If A >= B then
Print the value of A
Else
Print the value of B
Stop

	.BEGIN		
	IN	A	-- Get values for A and B
	IN	B	
	LOAD	B	
	COMPARE	A	-- If A >= 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

Program			Location Counter	Bindings	
	.BEGIN			Labels	addr's
Loop:	IN	X	0	Loop	0
	LOAD	X	1	Done	5
	COMPARE	Y	2	X	7
	JUMPLT	Done	3		
	JUMP	Loop	4		
Done :	OUT	Y	5		
	HALT		6		
X:	.DATA	0	7		
	.END				

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.