Computer Architecture

The von Neumann Architecture

- Programs are stored in memory along with the data they manipulate.
  - Allows the computer to be “multipurpose”, since the function can be changed by loading different program code.
- Dan’s Simple 4-bit Machine:
  - Stores binary data
  - Memory is 16 bytes long
  - Addresses are 4-bits long (decimal addresses 0 – 15)

The von Neumann Architecture

- Dan's Simple 4-bit Machine:
  - Some special "registers"
    - IP – Instruction Pointer (AKA PC – Program Counter)
    - IR – Instruction Register (contains the current instruction that is to be decoded and executed)
    - R0, R1 – Registers used for computation
  - Basic operation:
    - "Fetch" – to load the current instruction
    - "Decode" – to determine what to do
    - "Execute" – to carry out the operation

Demonstration: Tracing Execution

- Instruction Set
  - As described in the readings and in the hand-out
- Lets take a look at a sample program and simulate its operation by hand-tracing...

Address	*Value*
0000	0000 1111
0001	0101 0001
0010	1100 0001
0011	1000 0001
0100	0110 0010
0101	1110 0000
0110	0000 0000
0111	0000 0000
1000	0000 0000
1001	0000 0000
1010	0000 0000
1011	0000 0000
1100	0000 0000
1101	0000 0000
1110	0000 0000
1111	0000 0010

Address	*Value*	*Operation*
0000	0000 1111	load r0, 1111
0001	0101 0001	value r1, 0001
0010	1100 0001	r0 = r0-r1
0011	1000 0001	jumpzero r0, 0101
0100	0110 0010	jump 0010
0101	1110 0000	halt
0110	0000 0000	
0111	0000 0000	
1000	0000 0000	
1001	0000 0000	
1010	0000 0000	
1011	0000 0000	
1100	0000 0000	
1101	0000 0000	
1110	0000 0000	
1111	0000 0010	data

Your Turn to Trace

- Form groups of 3 – 4:
  - Someone to be “Recorder” (Track the changing state of the machine…)
  - Someone to be “Group Leader” (To keep the group on task…)
  - Team members:
    - Take turns decoding machine instructions in the IR
- Time:
  - 10 Minutes, then report to the class at large.

What is the memory state after this program has run with an initial IP value of 0000 0000 ?

Address	Value
0000	0000 1101
0001	0001 1111
0010	1010 0101
0011	0011 1111
0100	0000 1110
0101	0101 0001
0110	1100 0001
0111	0010 1110
1000	1000 1010
1001	0110 0000
1010	1110 0000
1011	0000 0000
1100	0000 0000
1101	0000 0011
1110	0000 0010
1111	0000 0000

Decode

Address	*Value*	*Operation*
0000	0000 1101	load r0, 1101
0001	0001 1111	load r1, 1111
0010	1010 0101	r1 = r0+r1
0011	0011 1111	store r1, 1111
0100	0000 1110	load r0, 1110
0101	0101 0001	r1 = 1
0110	1100 0001	r0=r0-r1
0111	0010 1110	store r0, 1110
1000	1000 1010	jumpzero r0, 1010
1001	0110 0000	jump 0000
1010	1110 0000	halt
1011	0000 0000	
1100	0000 0000	
1101	0000 0011	data
1110	0000 0010	data
1111	0000 0000	data

Result

Address	*Value*	*Operation*
0000	0000 1101	load r0, 1101
0001	0001 1111	load r1, 1111
0010	1010 0101	r1 = r0+r1
0011	0011 1111	store r1, 1111
0100	0000 1110	load r0, 1110
0101	0101 0001	r1 = 1
0110	1100 0001	r0=r0-r1
0111	0010 1110	store r0, 1110
1000	1000 1010	jumpzero r0, 1010
1001	0110 0000	jump 0000
1010	1110 0000	halt
1011	0000 0000	
1100	0000 0000	
1101	0000 0011	data
1110	0000 0000	data
1111	0000 0110	data

Writing Your Own Machine Language Program

- Form groups of 3 – 4:
  - Someone to be "Recorder" (Track the changing state of the machine...)
  - Someone to be "Group Leader" (To keep the group on task...)
  - Team members:
    - Think of the program first in English, then
    - Write the machine operations, then
    - Encode the machine operations into binary in memory

Writing Your Own Machine Language Program

- Algorithm:
  - A number N is the value initially stored in memory location 1111.
  - Calculate the sum of the first N integers.
  - Store the final sum in memory location 1110.
- Time:
  - 10 Minutes, then report to the class at large.

Write a program that will put in memory address 1110 the sum of the first N integers where N is the value stored in memory address 1111

Hint: Write the algorithm in English first, then translate it.

Address	*Value*
0000	
0001	
0010	
0011	
0100	
0101	
0110	
0111	
1000	
1001	
1010	
1011	
1100	
1101	
1110	0000 0000
1111	

Solution...
Does this work?

Address	*Value*	*Operation*
0000	0000 1110	load r0, 1110
0001	0101 0001	value r1, 0001
0010	1010 0001	r0=r0+r1
0011	0010 1110	store r0, 1110
0100	0000 1111	load r0, 1111
0101	1100 0001	r0=r0-r1
0110	1000 0000	jumpzero r0, 0000
0111	1110 0000	halt
1000	0000 0000	
1001	0000 0000	
1010	0000 0000	
1011	0000 0000	
1100	0000 0000	
1101	0000 0000	
1110	0000 0000	data
1111	xxxx xxxx	data

Debrief

- There is a LOT going on "under the hood" when a computer is running a program!
  - A GREAT DEAL of work involved:
    - To figure out how to lay out the algorithm,
    - To translate the algorithm into (human-readable) machine instructions, and then
    - To translate machine instructions into binary code.
    - To make sure that the program actually works!
- Possible exam question:
  - Given the instruction set and
  - A memory state,
    - Determine the ending state of the program.

Debrief

- We need some tools to make this less tedious!
  - That is why we have “high-level” languages like C++
    - To hide a lot of the “low level” details of the machine’s operation.
    - To allow us to write programs that can be reused on computers that have different architectures and instruction sets.