Binary and Hexadecimal Numbers

- Control Signals
  - The lowest level of our computer hardware works by turning billions of tiny switches (called transistors) on and off.
  - The ways that these switches are organised into sophisticated functional components is covered in CS1110.
  - For now, we'll abstract away from these details and recognise that these switches have two distinct states – they are either off or they are on.
  - We represent the off state by a 0 and the on state by a 1 and we can construct sequences of 1s and 0s to represent multiple control signals.
    - That is, we can use sequences of 1s and 0s to control our machine.
    - 101110 for example, might result in our machine turning on the memory and fetching some data.
    - In the appropriate context, these control signals act like instructions.
- A sequence of 1s and 0s can be interpreted as a number in Base 2

101

Binary and Hexadecimal Numbers

- What is a Number?
  - A number is a symbol, or a collection of symbols
  - The "number" of distinct symbols used determines the base.
  - The symbols in the base are ordered
  - The position of a symbols in a collection of symbols reflects the magnitude of that symbol in the collection
  - In the Decimal number system
    - There are 10 distinct symbols, orders as: 0,1,2,3,4,5,6,7,8,9
    - 5 is a number
    - 55 is a number, but the leftmost symbol has a greater magnitude than the rightmost

102

Binary and Hexadecimal Numbers

- In the Binary Number System,
  - There are 2 distinct symbols, orders as: 0,1
    - 1 is a number
    - 10 is a number, but the leftmost symbol has a greater magnitude than the rightmost
- In the Hexadecimal Number System,
  - There are 16 distinct symbols, orders as: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
    - 8 is a number
    - 3F is a number, but the leftmost symbol has a greater magnitude than the rightmost
- In a positional number system, the position of a symbol in a number determines its magnitude.
- Counting order determines how the symbols in a number system are used to represent more and more information.

Binary and Hexadecimal Numbers

- All number systems are equivalent, in that you can do the same things with all of them.
- Numbers in any base are the same in counting order:

Binary	Decimal	Hex
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Binary and Hexadecimal Numbers

- Note that, the smaller the base, the larger the sequence of symbols needed to represent the same information.
  - There is a trade-off between number of distinct symbols being manipulated and the size of sequences needed.
  - 1111 = 15 = F
- That said, specific number systems are easier to work with in specific situations.
  - Humans find it hard to work with large binary numbers
  - Machine are more complicated when they have to manipulate more that 2 states

105

Binary and Hexadecimal Numbers

- Data
  - We can use numbers together with specific interpretations to represent (model) any type of information
    - Characters are represented by their position in a standard table
    - Sound can be modelled by numbers derived from sampling analogue waveforms
    - Images can be digitized by CMOS cameras
    - Etc
- Therefore, apart from control signals and instructions, we can use binary numbers to represent and manipulate data in our computer.

106

Binary and Hexadecimal Numbers

- As already mentioned, binary is appropriate for the machine, but large sequences of binary numbers are hard for humans to differentiate.
- To address this, we can break the binary sequence into groups of 4 bits and represent then as a single hex digit.
- We can do this since any possible combination of 4 bits can be represented by a single hex digit.

Binary	Decimal	Hex
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

1010000011111100 = 50FC

107

Binary and Hexadecimal Numbers

- Therefore, sequences of hex digital can be used to represent the sequence of binary digits – which in turn represent control signal, instructions and data in our computer.

108

From Hex to Mnemonics

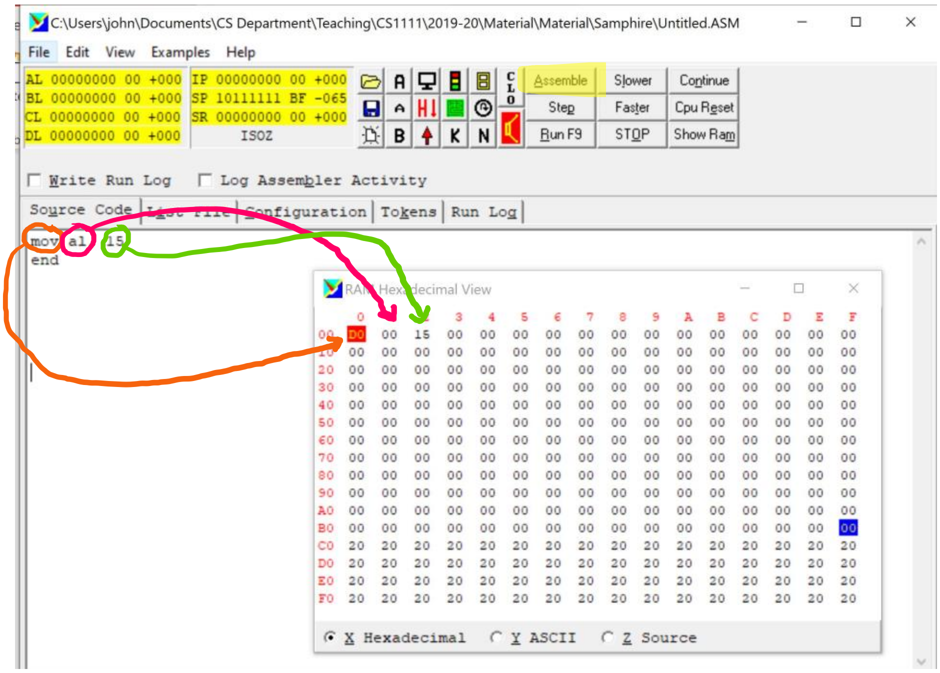
- From Hex to Assembly mnemonics
  - By design, the sequence of hex digits representing instructions of a program will consist of groups of hex numbers – each group representing a distinct instruction.
    - The design of these groups is done as part of the Instruction Set Architecture Level.
  - In general, the first hex number will represent the operation to be performed (the Opcode) and subsequent hex numbers (if any) will represent operand data or the location of operand data.
  - In the Assembly Language, the hex Opcode is given a name to describe the operation, the data may appear as a literal hex number and the location of the operand data may be a hex number referring to a register (containing the data or the address of the data, for example)

109

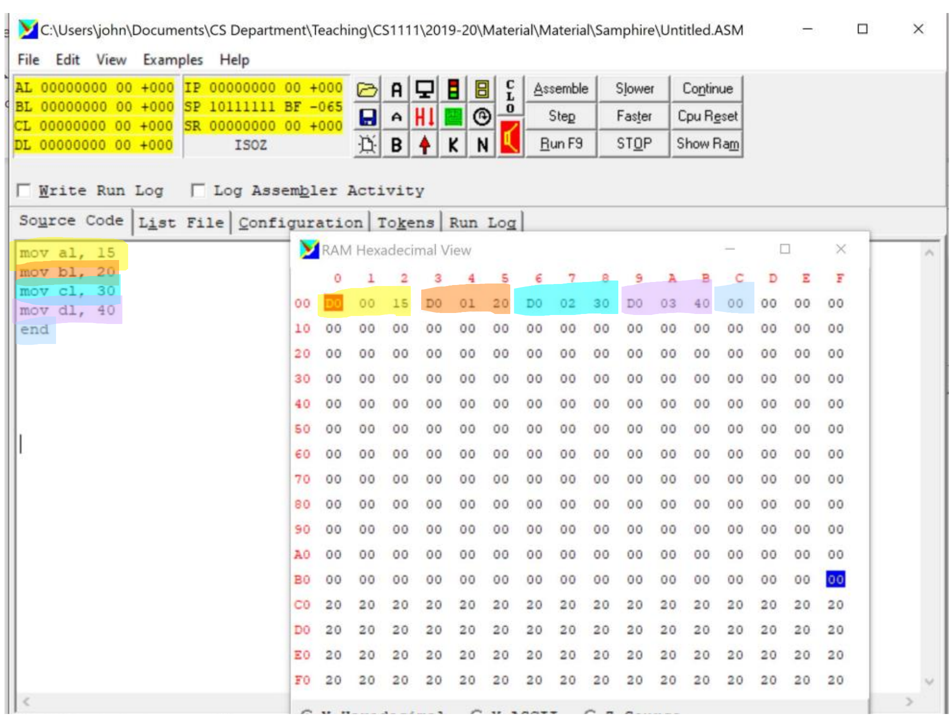
From Hex to Mnemonics

- An illustration using Samphire
  - Consider the binary instruction sequence:
    - 110100000000000000011010
    - This instructs the machine to put the hex number 15 into the al register
    - As a hex sequence it is: D00015
      - D0 is the Opcode
      - 00 is the identification of the al register
      - 15 is a hex constant
    - As an assembly language instruction it is:
      - mov al, 15
  - The Assembler performs these steps in reverse

110


C:\Users\john\Documents\CS Department\Teaching\CS1111\2019-20\Material\Material\Samphire\Untitled.ASM
File Edit View Examples Help
AL 00000000 00 +000
BL 00000000 00 +000
CL 00000000 00 +000
DL 00000000 00 +000
IP 00000000 00 +000
SP 10111111 BF -065
SR 00000000 00 +000
ISOZ
Assemble
Slower
Continue
Step
Faster
Cpu Reset
Run F9
STOP
Show Ram
Write Run Log
Log Assembler Activity
Source Code
Configuration
Tokens
Run Log
mov al, 15
end
RAM Hexadecimal View
00 D0 00 15 00 00 00 00 00 00 00 00 00 00 00 00 00
X Hexadecimal
Y ASCII
Z Source

111


C:\Users\john\Documents\CS Department\Teaching\CS1111\2019-20\Material\Material\Samphire\Untitled.ASM
File Edit View Examples Help
AL 00000000 00 +000
BL 00000000 00 +000
CL 00000000 00 +000
DL 00000000 00 +000
IP 00000000 00 +000
SP 10111111 BF -065
SR 00000000 00 +000
ISOZ
Assemble
Slower
Continue
Step
Faster
Cpu Reset
Run F9
STOP
Show Ram
Write Run Log
Log Assembler Activity
Source Code
List File
Configuration
Tokens
Run Log
mov al, 15
mov bl, 20
mov cl, 30
mov dl, 40
end
RAM Hexadecimal View
00 D0 00 15 D0 01 20 D0 02 30 D0 03 40 00 00 00 00

112

Action of the Samphire Assembler

- The Assembler translates each assembly language instruction, in turn, and loads the resulting machine code into consecutive locations in memory, starting at address 0.
  - Each memory location can hold 1 byte of data (i.e., 2 hex digits)
- Each of the data registers: al, bl, cl and dl is given a unique identifier: 00, 01, 02, 03, respectively
- Not all instructions are the same size
  - mov al, 15 is 3 bytes long
  - end is 1 byte long
- The Instruction Pointer (IP) is initialized to the address of the first instruction.
- We will see other actions of the Assembler later when we examine different instructions.

113

Program Execution

- When the run (or step) button is pressed, the simulator starts the fetch-decode-execute cycle: fetching, decoding and executing instructions in memory until the end instruction is executed. The end instruction terminates the fetch-decode-execute cycle

114

Sample Programs

- Sample programs introducing the instructions of the assembly language are presented.
- Move instructions and addressing modes.
  - Address modes refer to the manner in which operands are accessed
  - Mov Immediate Instructions
    - The operand to be moved to a destination register appears literally in the instruction
    - mov al, 15
  - Register Moves
    - The operand to be moved to a destination register is in another register
    - mov al, bl – not possible in Samphire
  - Indirect Moves
    - The memory address of the operand to be moved, or the memory address of the destination for the operand appears literally in the instruction
    - mov al, [80]
    - mov [85], al
  - Register Indirect Moves
  - The memory address of the operand to be moved, or the memory address of the destination for the operand is in a register.
    - mov al, [cl]
    - mov [cl], al

115