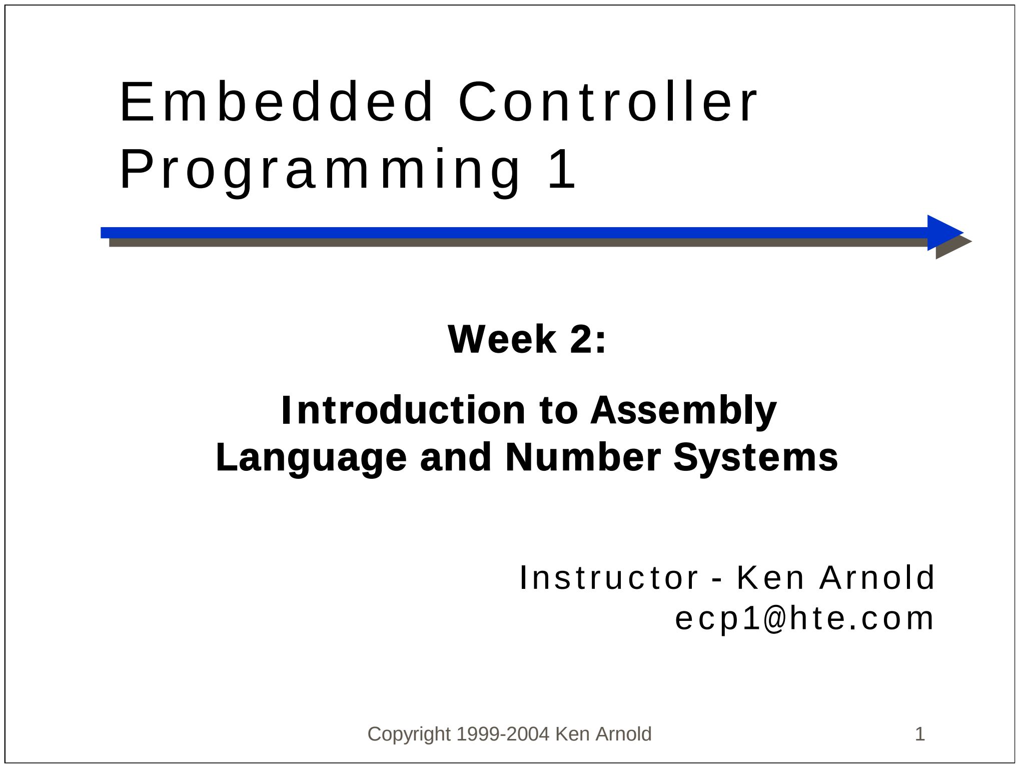


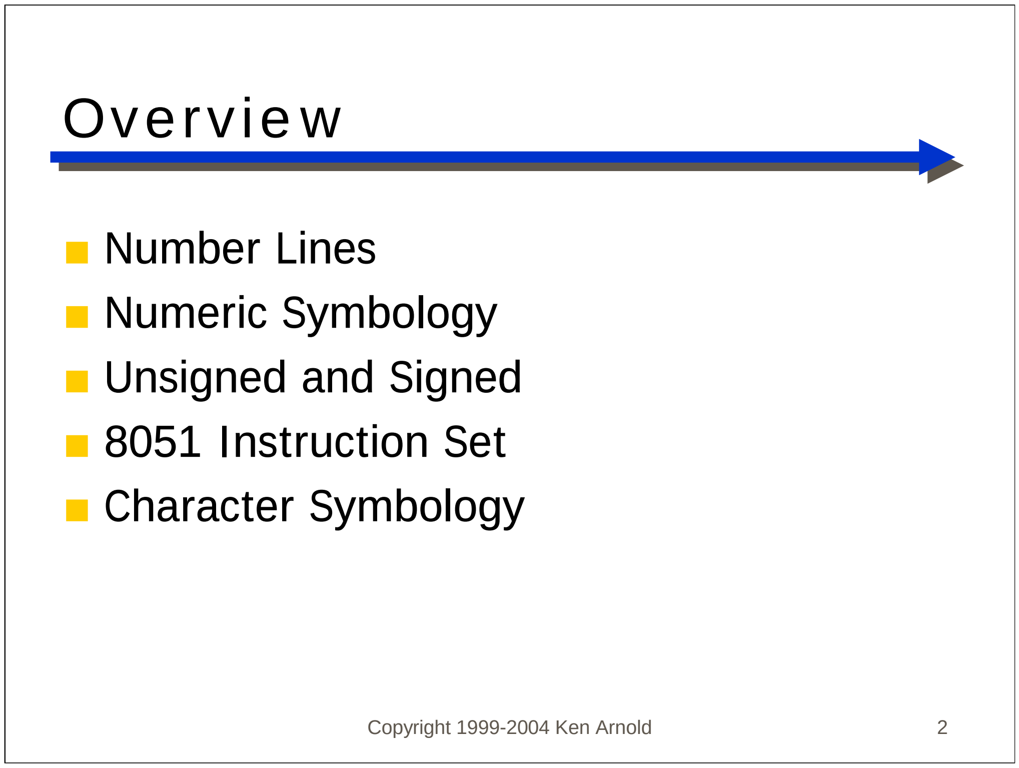
Embedded Controller Programming 1

Week 2:

Introduction to Assembly Language and Number Systems

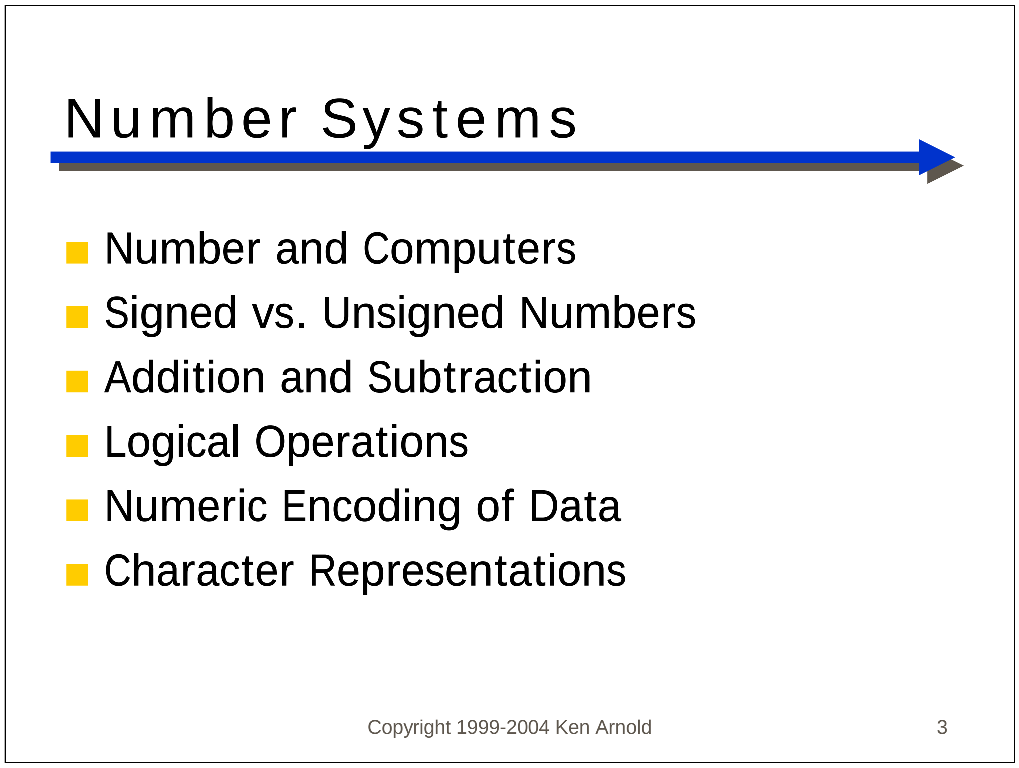
Instructor - Ken Arnold
ecp1@hte.com

1



Overview

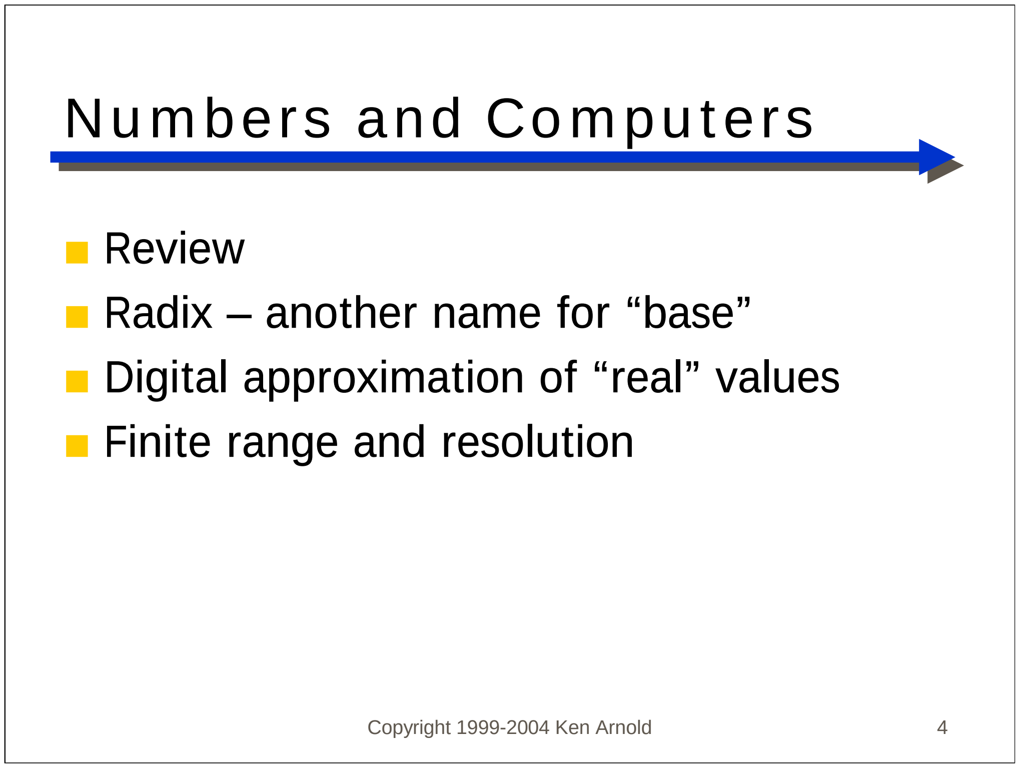
- Number Lines
- Numeric Symbology
- Unsigned and Signed
- 8051 Instruction Set
- Character Symbology



Number Systems

- Number and Computers
- Signed vs. Unsigned Numbers
- Addition and Subtraction
- Logical Operations
- Numeric Encoding of Data
- Character Representations

3

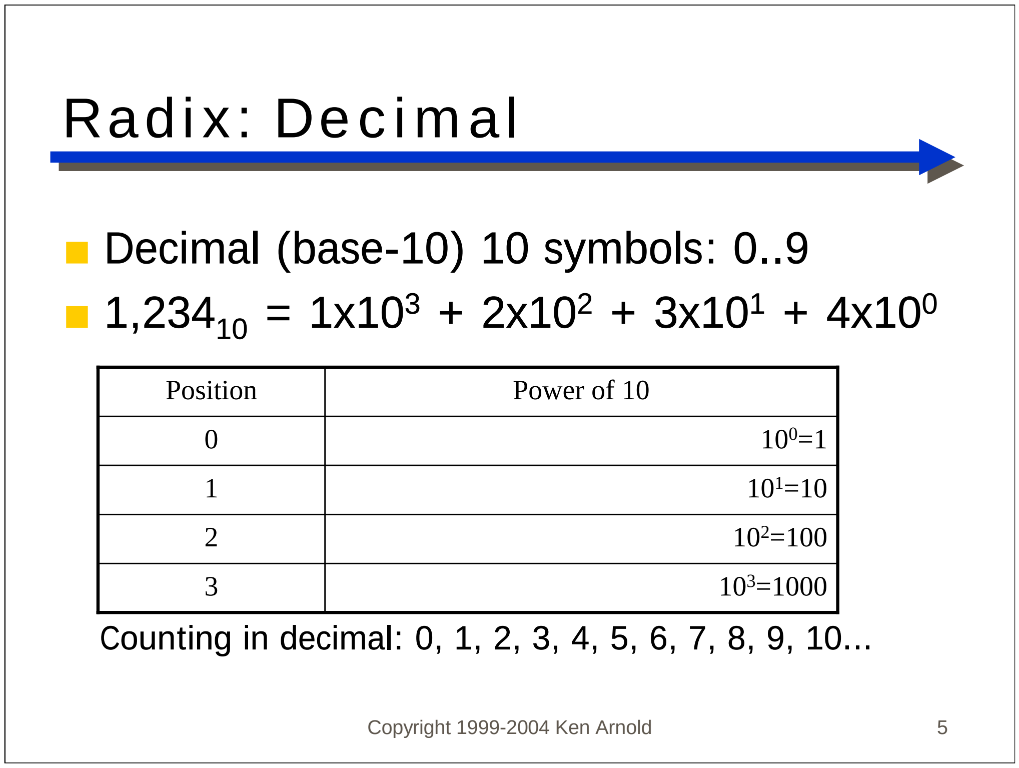


Numbers and Computers

- Review
- Radix – another name for “base”
- Digital approximation of “real” values
- Finite range and resolution

•Although humans tend to communicate symbolically, it is important to realize, and I mean truly understand, that computers are numeric processing engines. Everything on a computer boils down to a number. This is true for the data used by your application and it is true for the instructions which comprise your application. You are already familiar with many different number systems: Metric and English measures, Centigrade and Fahrenheit, if fly or boat you understand knots, furlongs, fortnights, Newtons, Calories…I think you get the idea. Computers simply present us with an opportunity to create new ways of representing numbers.

•Since childhood you have been learning to count in a number of different radices: radix of 10 (decimal) when you count, radix of 12 when you measure things in inches and feet, radix of 60 when you measure in seconds and minutes. Computers use a radix of 2 (binary) for everything. In the “old” days, we represented numbers in octal,which is a radix of 8. This was convenient because it is very easy to map binary to and from octal values. Nowadays, almost everything is done in hexadecimal (radix of 16). Hexadecimal results in a more compact number representation, and also maps very easily to and from binary.

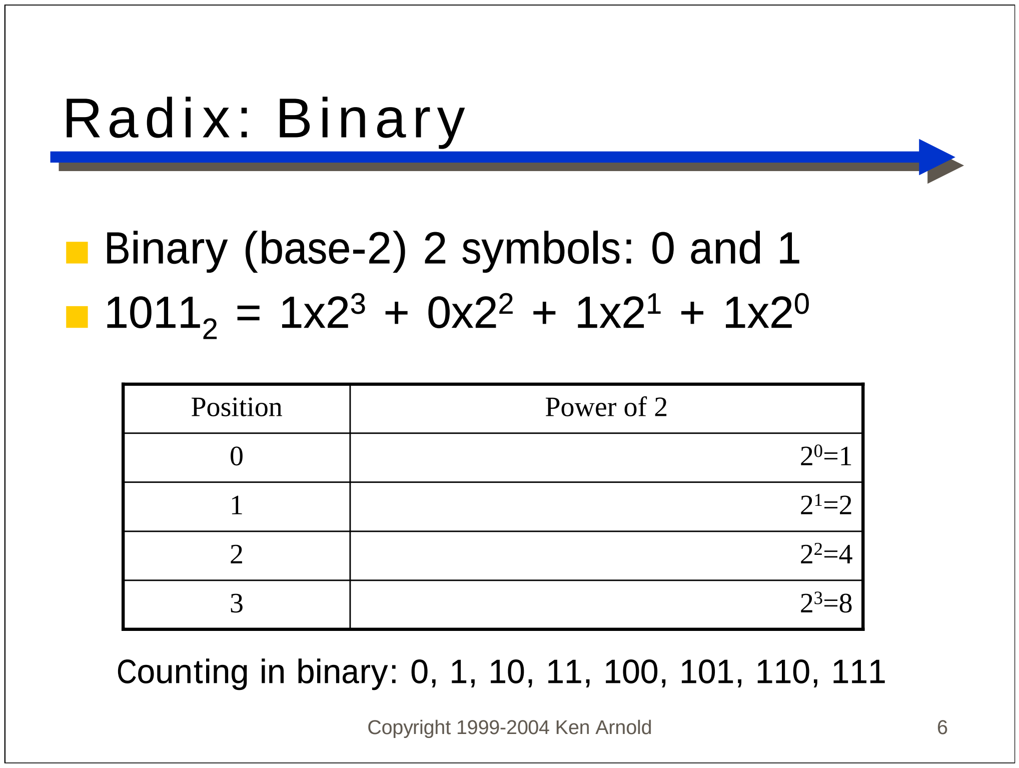


Radix: Decimal

- Decimal (base-10) 10 symbols: 0..9
- $1{,}234_{10} = 1\text{x}10^3 + 2\text{x}10^2 + 3\text{x}10^1 + 4\text{x}10^0$

Position	Power of 10
0	$10^0=1$
1	$10^1=10$
2	$10^2=100$
3	$10^3=1000$

Counting in decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10...



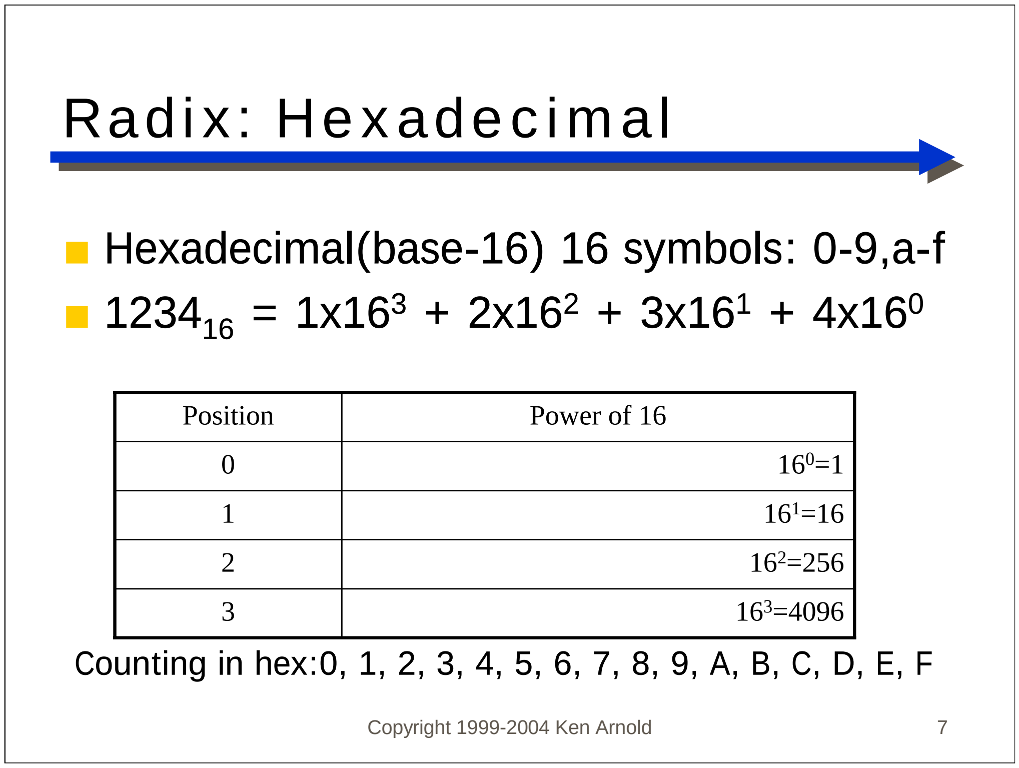
Radix: Binary

- Binary (base-2) 2 symbols: 0 and 1
- $1011_2 = 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$

Position	Power of 2
0	$2^0=1$
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$

Counting in binary: 0, 1, 10, 11, 100, 101, 110, 111

6



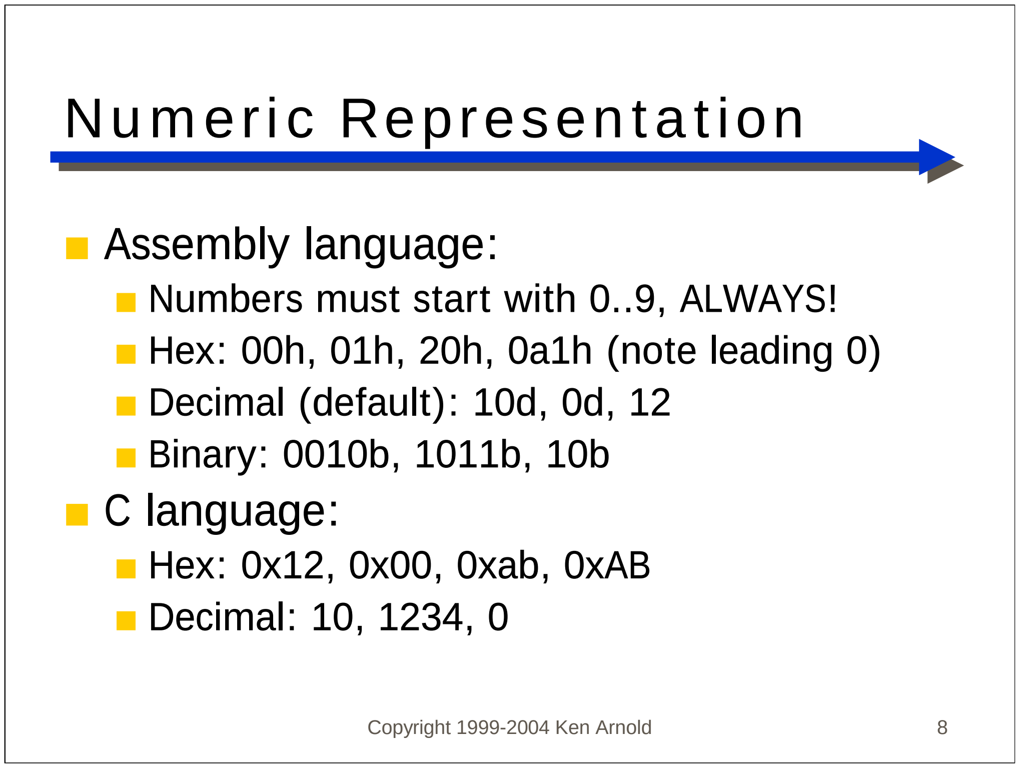
Radix: Hexadecimal

- Hexadecimal(base-16) 16 symbols: 0-9,a-f
- $1234_{16} = 1x16^3 + 2x16^2 + 3x16^1 + 4x16^0$

Position	Power of 16
0	$16^0=1$
1	$16^1=16$
2	$16^2=256$
3	$16^3=4096$

Counting in hex:0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Copyright 1999-2004 Ken Arnold 7



Numeric Representation

- Assembly language:
  - Numbers must start with 0..9, ALWAYS!
  - Hex: 00h, 01h, 20h, 0a1h (note leading 0)
  - Decimal (default): 10d, 0d, 12
  - Binary: 0010b, 1011b, 10b
- C language:
  - Hex: 0x12, 0x00, 0xab, 0xAB
  - Decimal: 10, 1234, 0

8

•It’s important to understand how to enter numeric values into your program. Examples for moving 22_{10} into the accumulator:

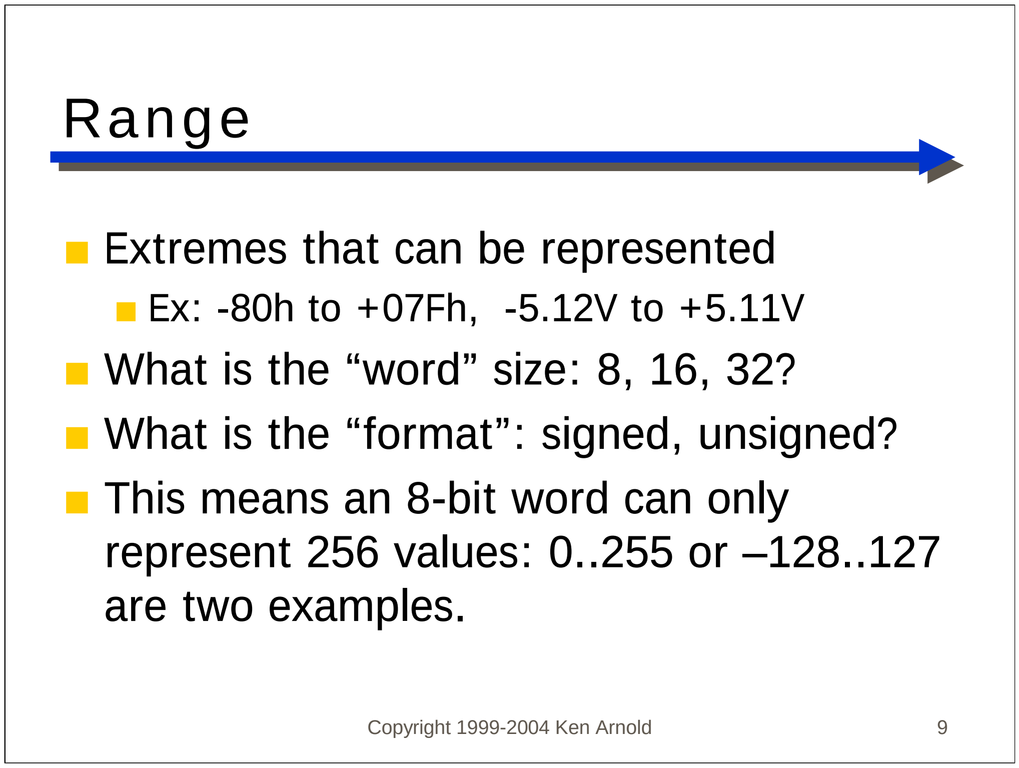
•MOV A,#16h

•MOV A,#22d

•In a high level language, such as C, we would write the following:

•ACC = 0x16;

•ACC = 22;

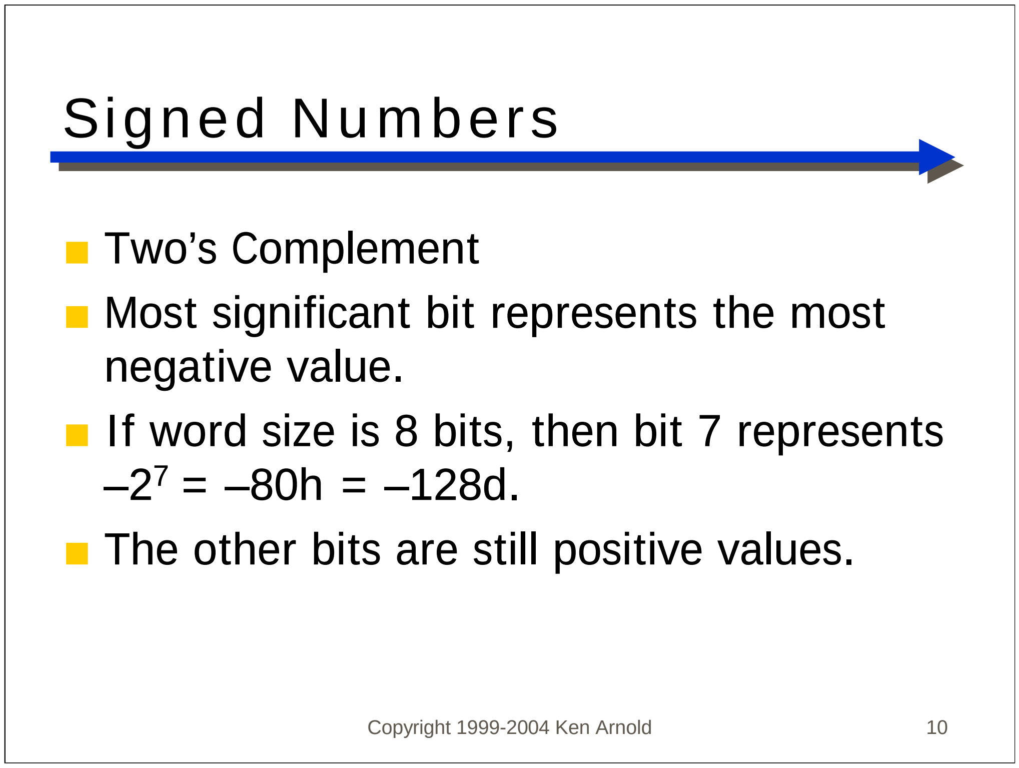


Range

- Extremes that can be represented
  - Ex: -80h to +07Fh, -5.12V to +5.11V
- What is the “word” size: 8, 16, 32?
- What is the “format”: signed, unsigned?
- This means an 8-bit word can only represent 256 values: 0..255 or –128..127 are two examples.

Copyright 1999-2004 Ken Arnold 9

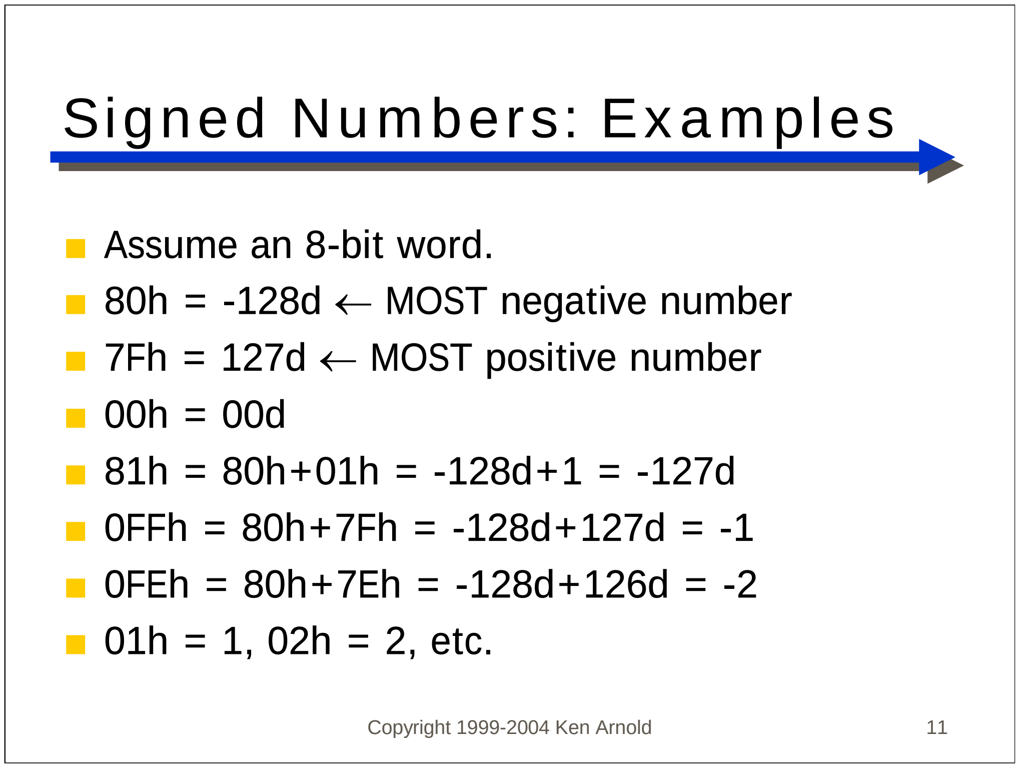
•The range of numbers which can be represented on a computer is limited by a number of factors, including the size of the storage area for the value and the format of the value. Say we only have 8 bits of storage for the value. If we decided that the value is unsigned, then we have a range from 0 to 0FFh. If the value must be signed (that is, it may have negative as well as positive values), then our range is from –80h to 07Fh because we must use one of the data bits to represent the sign of the value.



Signed Numbers

- Two’s Complement
- Most significant bit represents the most negative value.
- If word size is 8 bits, then bit 7 represents $-2^7 = -80h = -128d$.
- The other bits are still positive values.

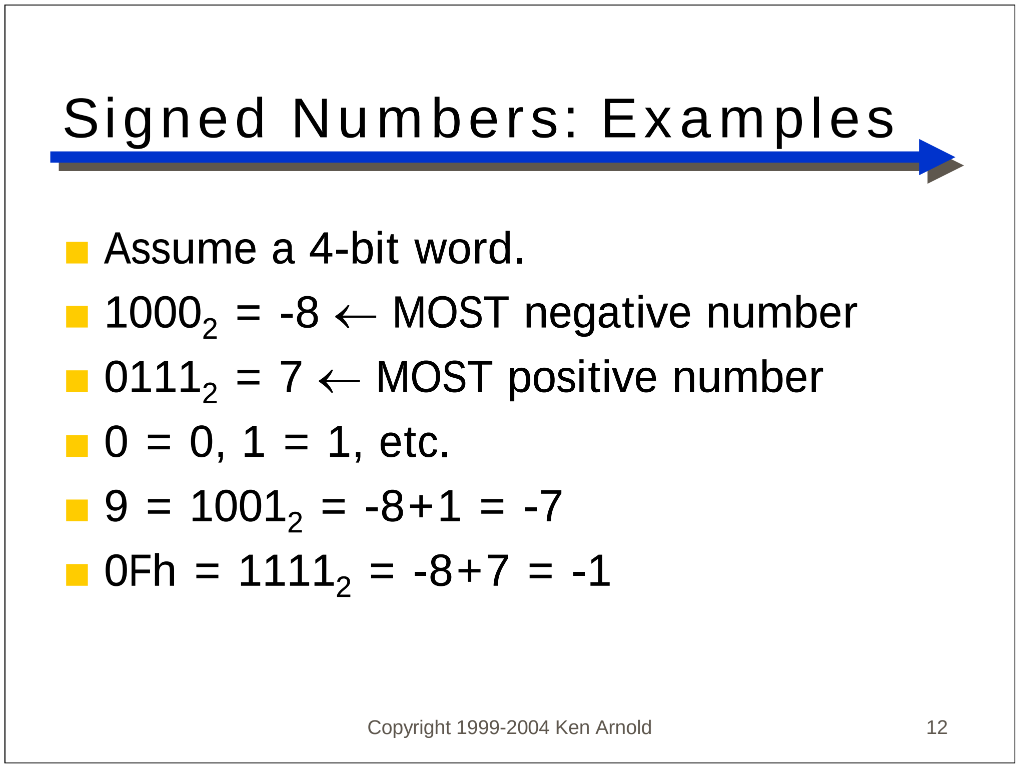
10



Signed Numbers: Examples

- Assume an 8-bit word.
- 80h = -128d ← MOST negative number
- 7Fh = 127d ← MOST positive number
- 00h = 00d
- 81h = 80h+01h = -128d+1 = -127d
- 0FFh = 80h+7Fh = -128d+127d = -1
- 0FEh = 80h+7Eh = -128d+126d = -2
- 01h = 1, 02h = 2, etc.

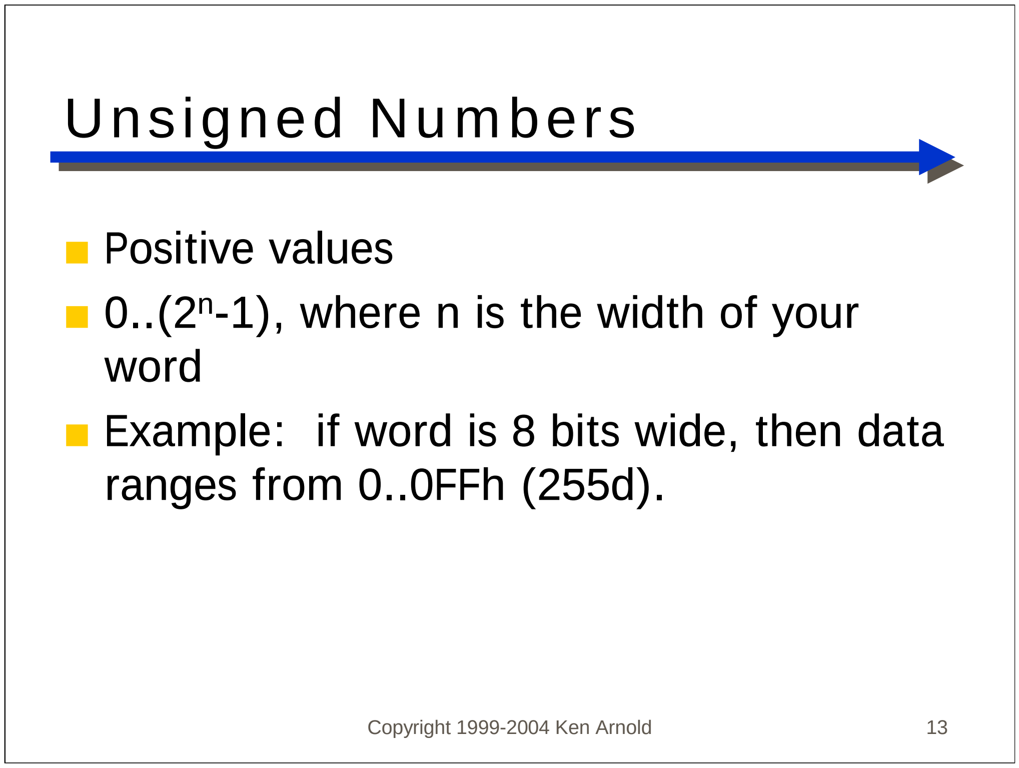
11



Signed Numbers: Examples

- Assume a 4-bit word.
- 1000_2 = -8 ← MOST negative number
- 0111_2 = 7 ← MOST positive number
- 0 = 0, 1 = 1, etc.
- 9 = 1001_2 = -8+1 = -7
- 0Fh = 1111_2 = -8+7 = -1

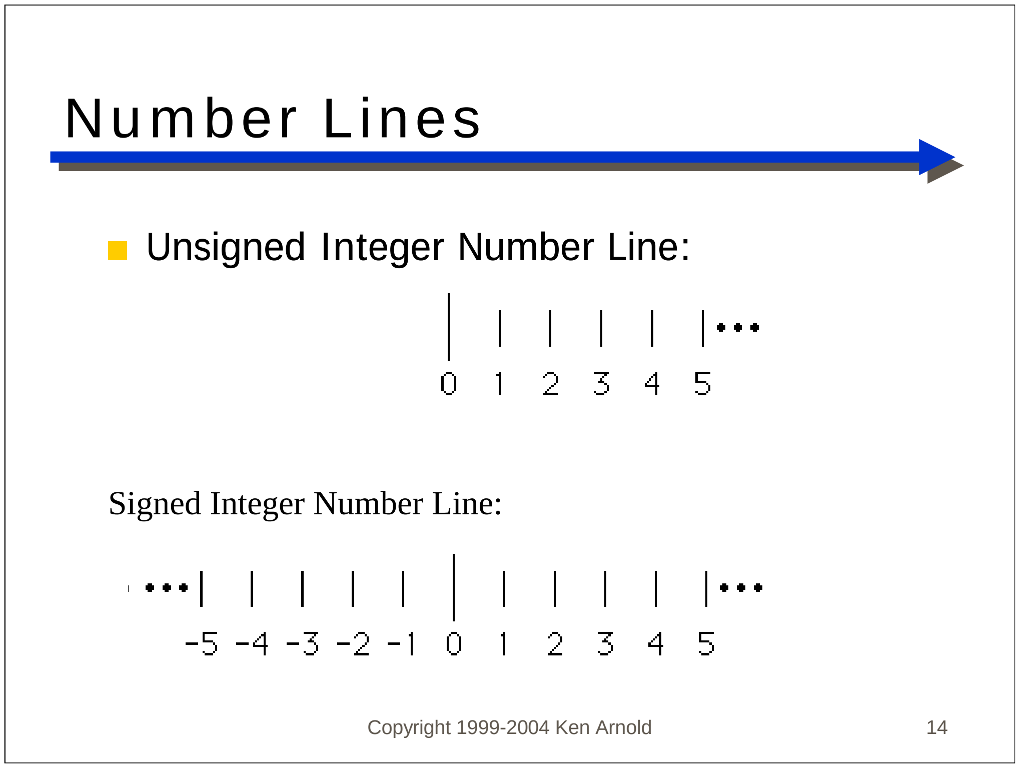
12



Unsigned Numbers

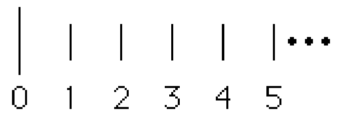
- Positive values
- 0..(2^n-1), where n is the width of your word
- Example: if word is 8 bits wide, then data ranges from 0..0FFh (255d).

13

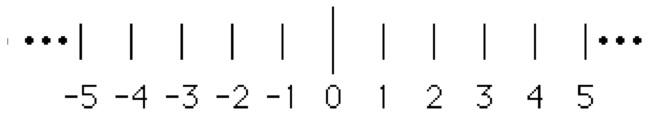


Number Lines

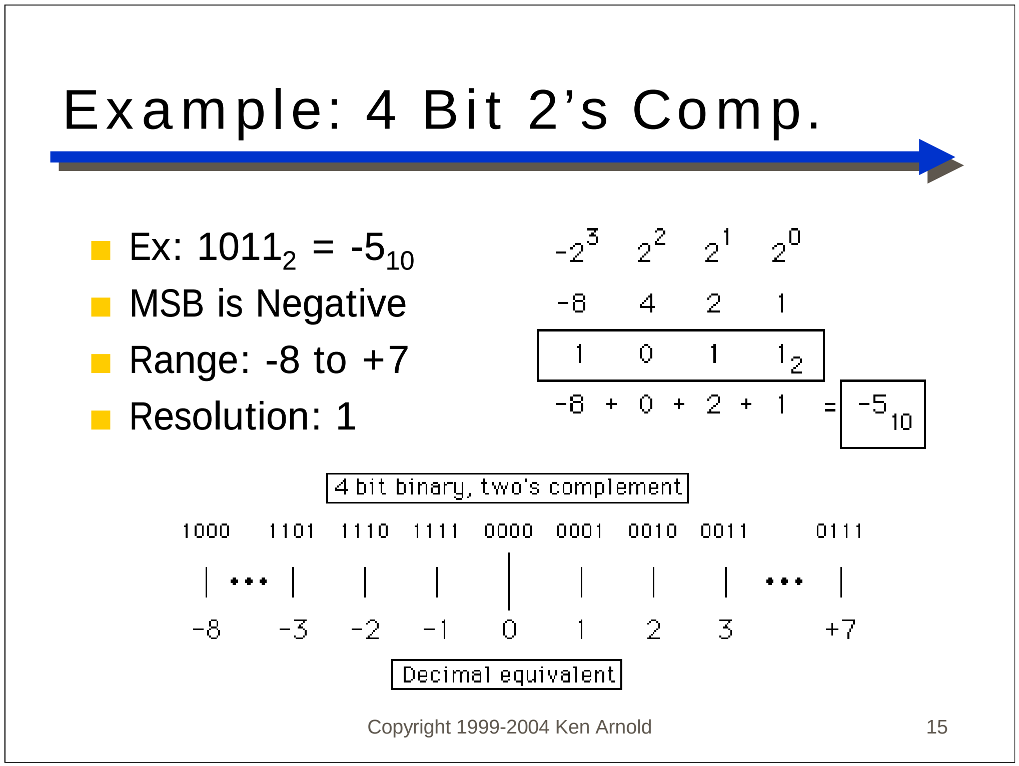
- Unsigned Integer Number Line:



Signed Integer Number Line:



Copyright 1999-2004 Ken Arnold 14



Example: 4 Bit 2's Comp.

- Ex: 1011_2 = -5_{10}
- MSB is Negative
- Range: -8 to +7
- Resolution: 1

-2^3	2^2	2^1	2^0
-8	4	2	1
1	0	1	1_2

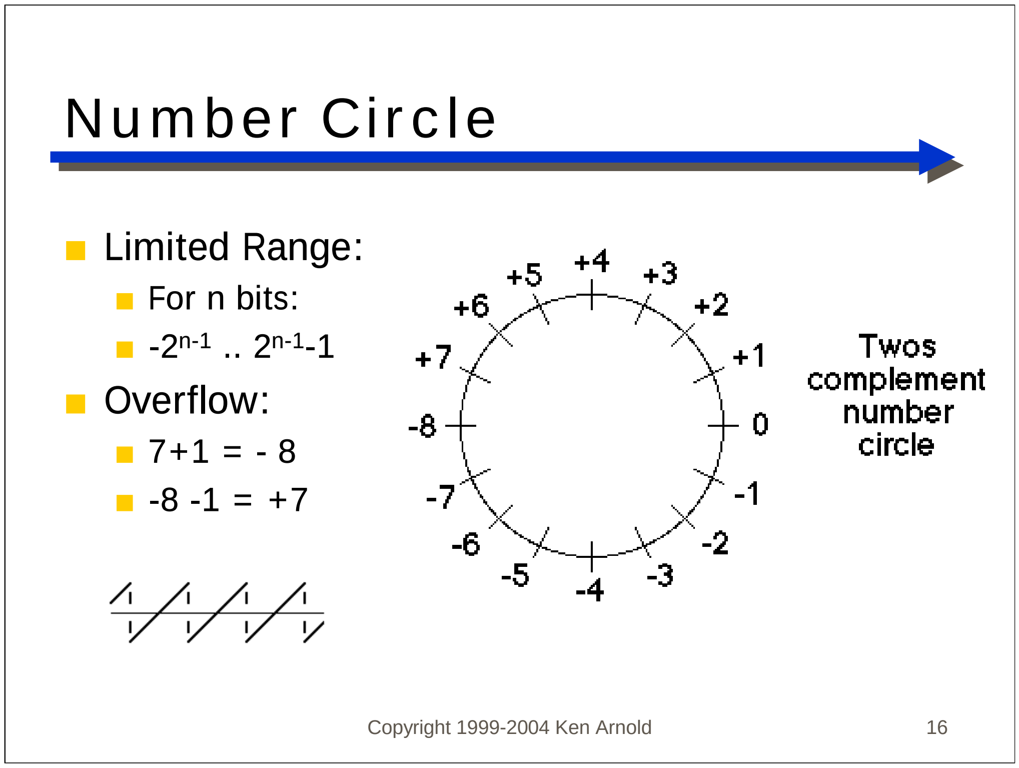
$-8 + 0 + 2 + 1 = \boxed{-5_{10}}$

4 bit binary, two's complement

1000	…	1101	1110	1111	0000	0001	0010	0011	…	0111
-8		-3	-2	-1	0	1	2	3		+7

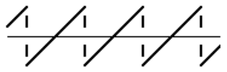
Decimal equivalent

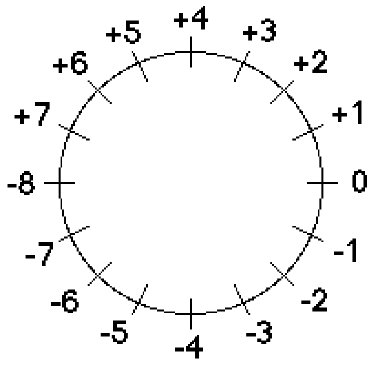
Copyright 1999-2004 Ken Arnold 15



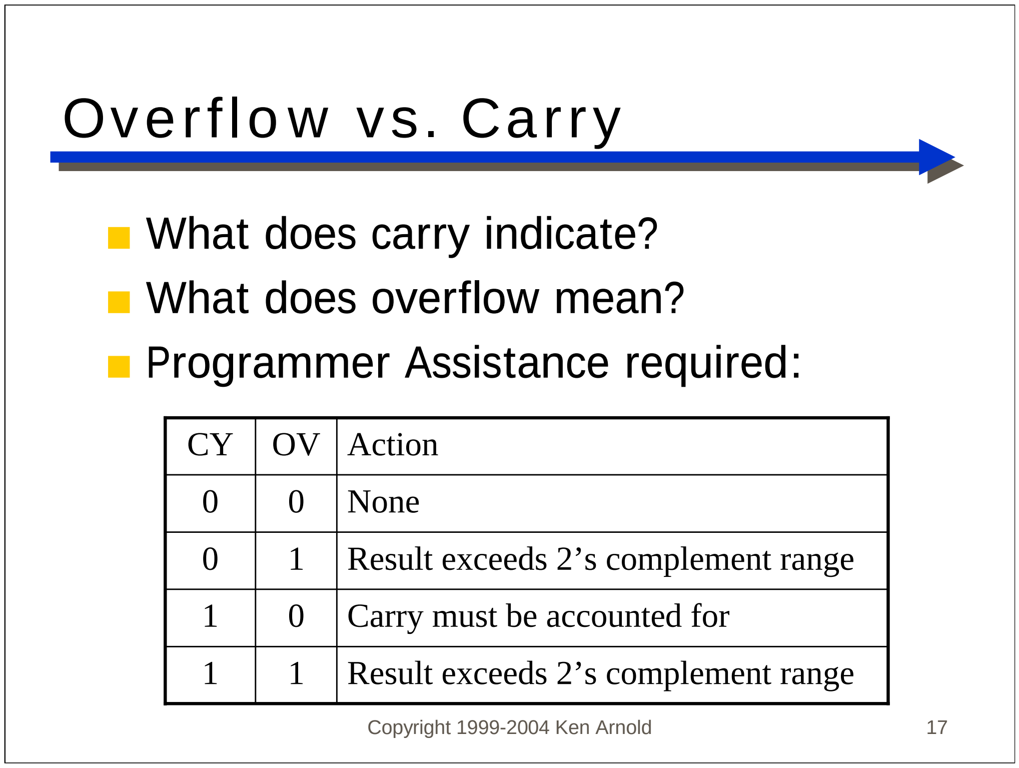
Number Circle

- Limited Range:
  - For n bits:
  - $-2^{n-1} .. 2^{n-1}-1$
- Overflow:
  - 7+1 = - 8
  - -8 -1 = +7





Twos complement number circle

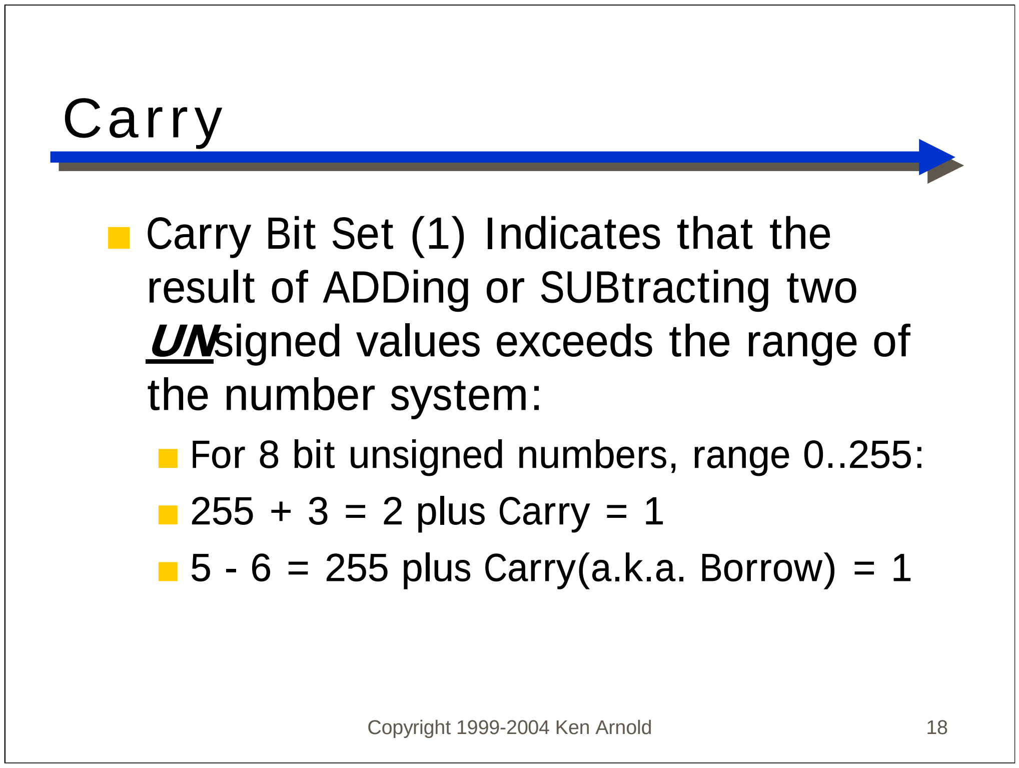


Overflow vs. Carry

- What does carry indicate?
- What does overflow mean?
- Programmer Assistance required:

CY	OV	Action
0	0	None
0	1	Result exceeds 2’s complement range
1	0	Carry must be accounted for
1	1	Result exceeds 2’s complement range

17

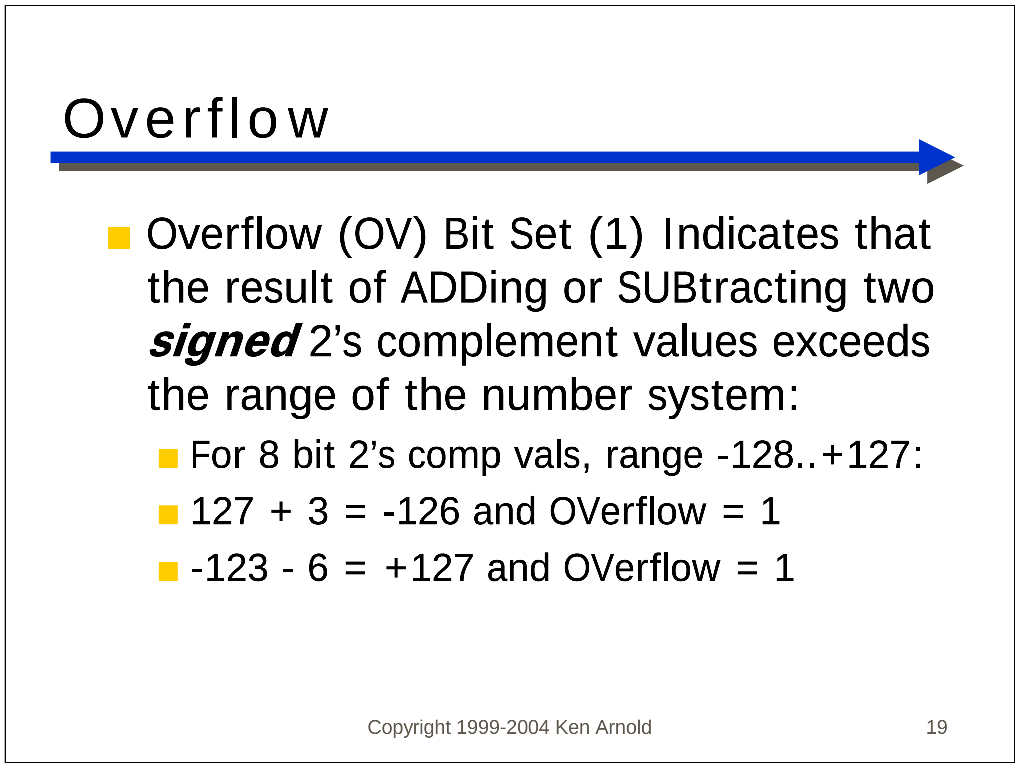


Carry

- Carry Bit Set (1) Indicates that the result of ADDing or SUBtracting two ***UN***signed values exceeds the range of the number system:
  - For 8 bit unsigned numbers, range 0..255:
  - 255 + 3 = 2 plus Carry = 1
  - 5 - 6 = 255 plus Carry(a.k.a. Borrow) = 1

18

•In terms of subtraction, the carry flag is set when a borrow is required. Again, the carry flag is used to “extend” the register size from 8 bits to a pretend 9 bits.



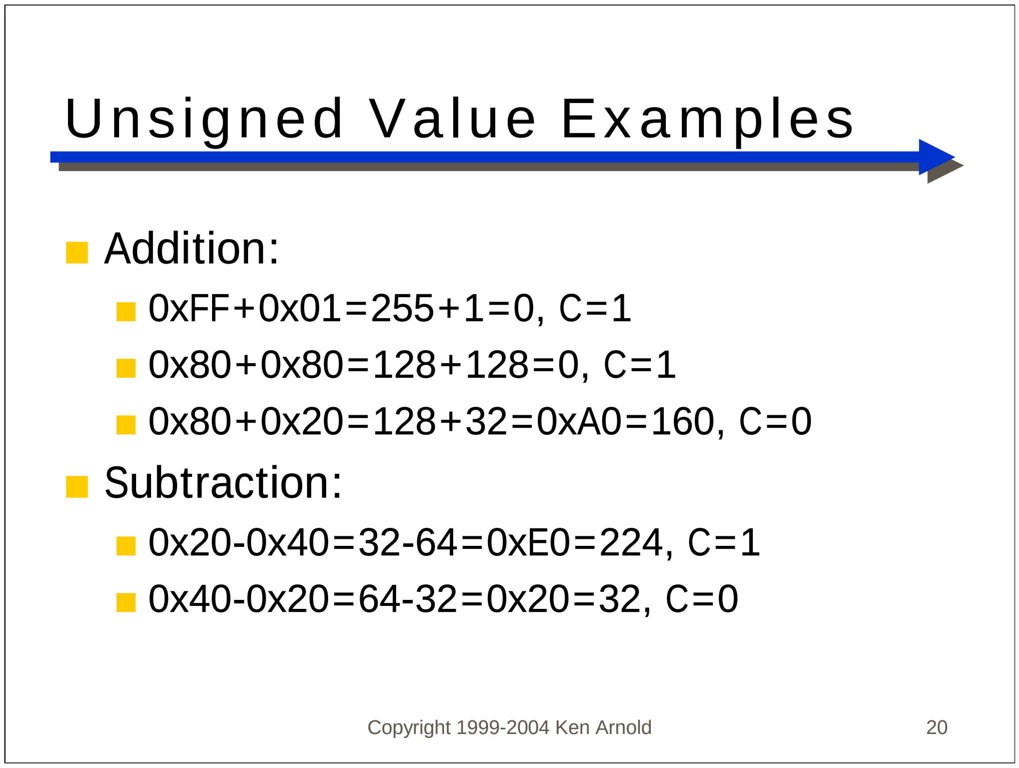
Overflow

- Overflow (OV) Bit Set (1) Indicates that the result of ADDing or SUBtracting two ***signed*** 2’s complement values exceeds the range of the number system:
  - For 8 bit 2’s comp vals, range -128..+127:
  - 127 + 3 = -126 and OVerflow = 1
  - -123 - 6 = +127 and OVerflow = 1

Copyright 1999-2004 Ken Arnold 19

•Overflow occurs when the result of adding or subtracting two signed, two’s complement numbers results in a number that cannot be represented with 8 bits. The sign of the result is wrong.

•An overflow condition is indicated when the Overflow flag (OV) is set. This flag is found in the PSW of the 8051, bit 2. This flag only has meaning during signed arithmetic operations. The overflow flag is set anytime you inadvertently and incorrectly change the sign of a result. The text expresses this as OV = C7 XOR C6. This means an overflow occurs if you have either a carry from bit 7 or a carry from bit 6, but not both.

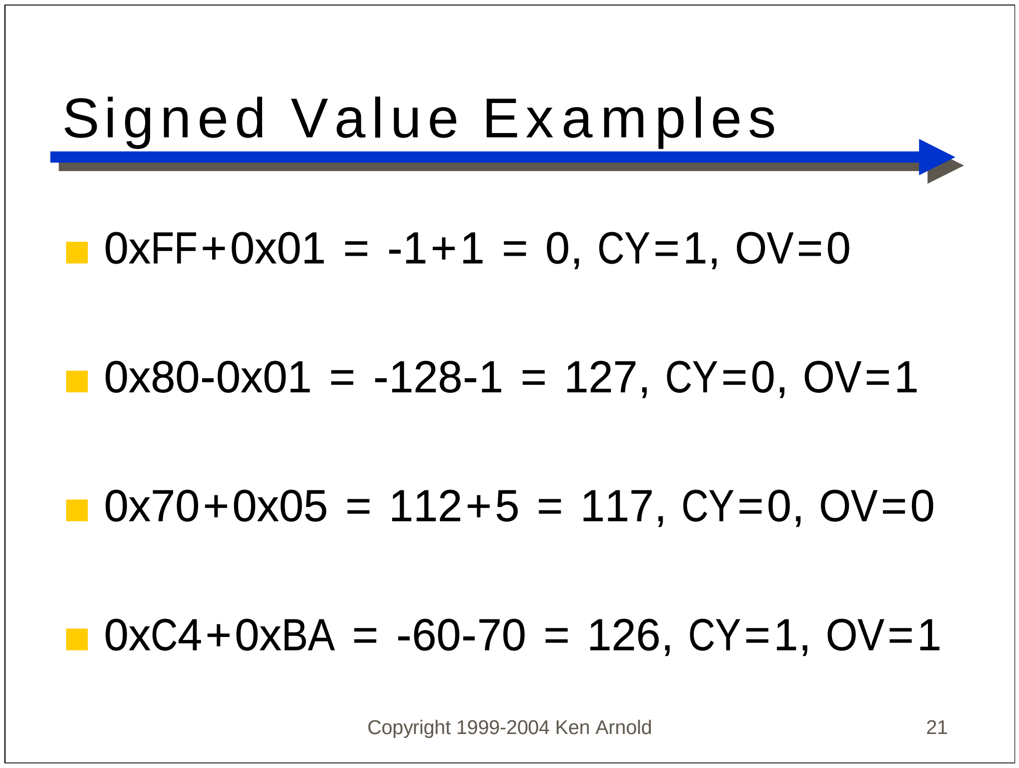


Unsigned Value Examples

- Addition:
  - 0xFF+0x01=255+1=0, C=1
  - 0x80+0x80=128+128=0, C=1
  - 0x80+0x20=128+32=0xA0=160, C=0
- Subtraction:
  - 0x20-0x40=32-64=0xE0=224, C=1
  - 0x40-0x20=64-32=0x20=32, C=0

Copyright 1999-2004 Ken Arnold 20

Carry is sort of like overflow for unsigned numbers. The result of an add or subtract won’t fit in an 8 bit unsigned result.



Signed Value Examples

- 0xFF+0x01 = -1+1 = 0, CY=1, OV=0
- 0x80-0x01 = -128-1 = 127, CY=0, OV=1
- 0x70+0x05 = 112+5 = 117, CY=0, OV=0
- 0xC4+0xBA = -60-70 = 126, CY=1, OV=1

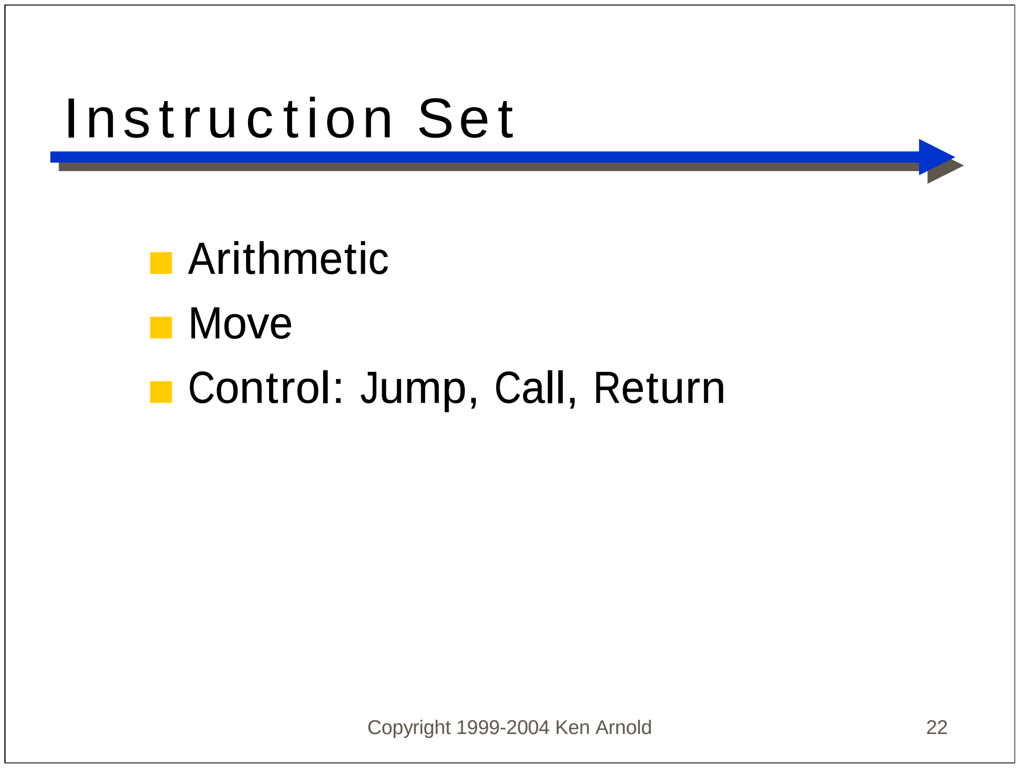
•In the first example, when we look at the binary we have $(1111\ 1111)_2$+$(0000\ 0001)_2$=$(1\ 0000\ 0000)_2$. You can see that the 9th bit is set, which is CY. However, because we had an overflow from both bit 6 and bit 7, there is no overflow.

•Again, let's fall back on binary for the next example: $(1000\ 0000)_2$-$(0000\ 0001)_2$=$(0111\ 1111)_2$. Here, there was not a borrow for bit 7 (no carry), but there is a borrow for bit 6…an overflow! Note how the sign is wrong and so programmer assistance is required to fix the sign bit.

•Normal addition, just like the unsigned version for this example.

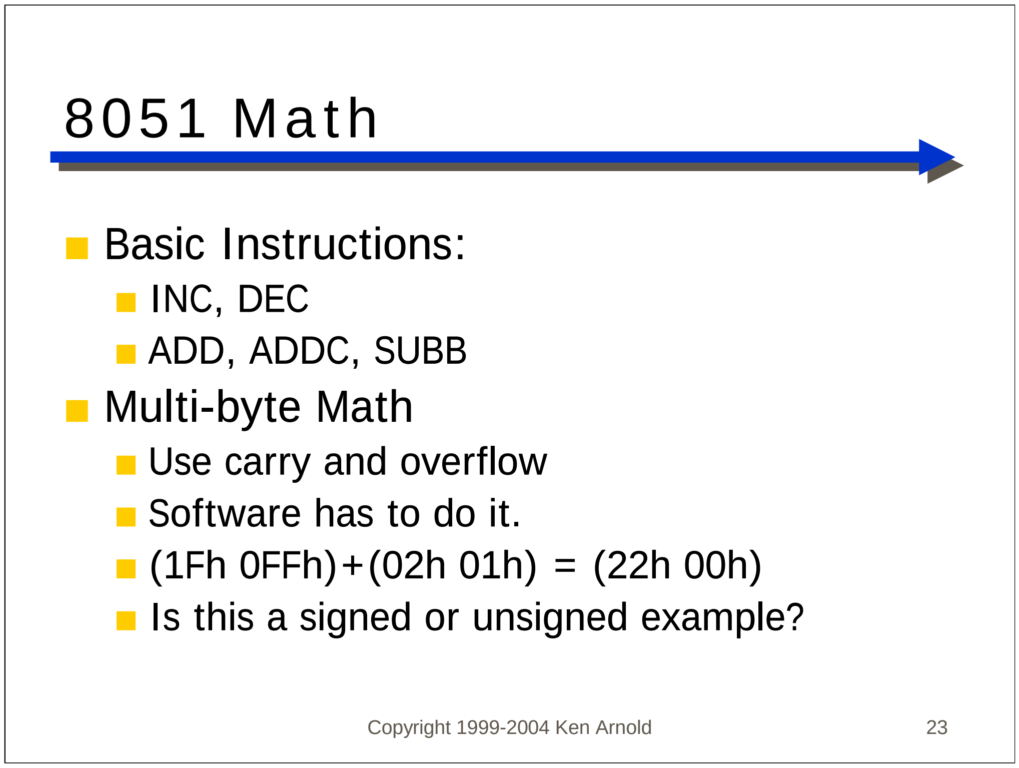
•Finally, an example with both the CY and OV bits set: $(1100\ 0100)_2$+$(1011\ 1100)_2$=$(1\ 0111\ 1110)_2$. There is a carry from bit 7, but no carry from bit 6, so we have a carry and overflow. Now you might say "-60-70 isn't 126 it's –130! Even if I change the sign, I just get –126. What's going on?" Well, you can't forget the fact that you have a range of –128..127. -130 is outside the range you can represent.

•Side note: In the above example, if you use the carry bit to be –256, then 126-256=-130.



Instruction Set

- Arithmetic
- Move
- Control: Jump, Call, Return



8051 Math

- Basic Instructions:
  - INC, DEC
  - ADD, ADDC, SUBB
- Multi-byte Math
  - Use carry and overflow
  - Software has to do it.
  - (1Fh 0FFh)+(02h 01h) = (22h 00h)
  - Is this a signed or unsigned example?

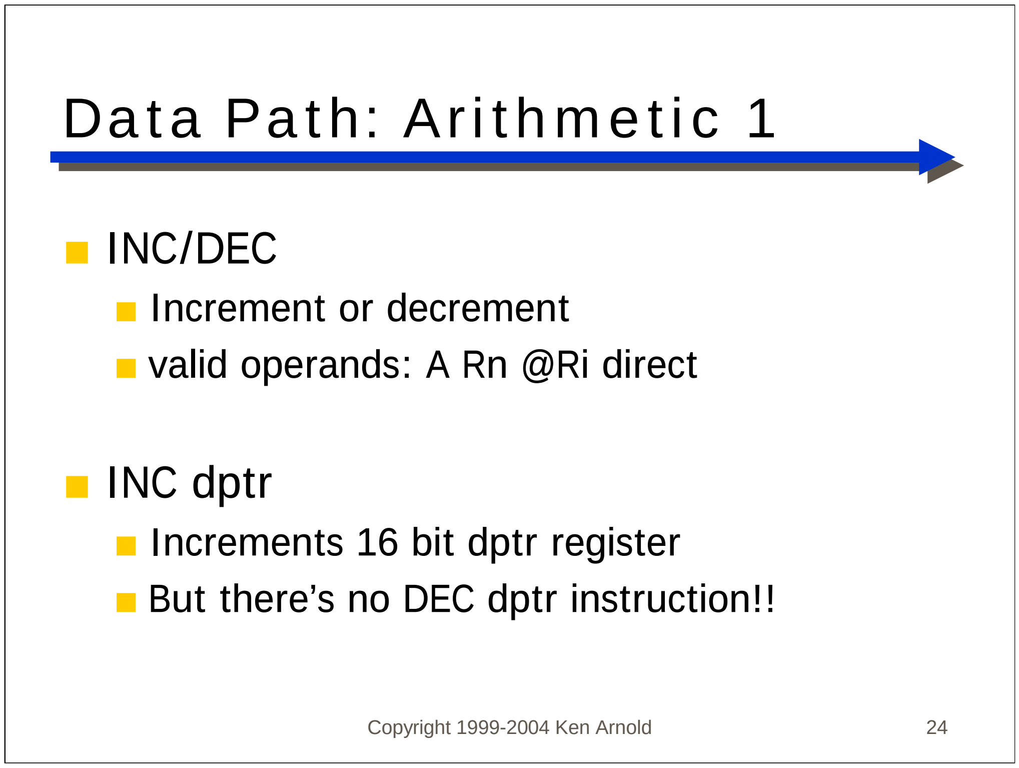
23

•The simplest operations are the increment and decrement operation, adding and subtracting by one. It is important to note that these operations do not affect the math flags (CY or OV). One more “feature” to keep in mind is that, while you can INC DPTR, there is not a matching DEC DPTR.

•The difference between the ADD and ADDC is that the ADD ignores the carry bit on input and the ADDC uses the carry bit as input. Keep in mind that the SUBB instruction always uses the carry bit as a borrow.

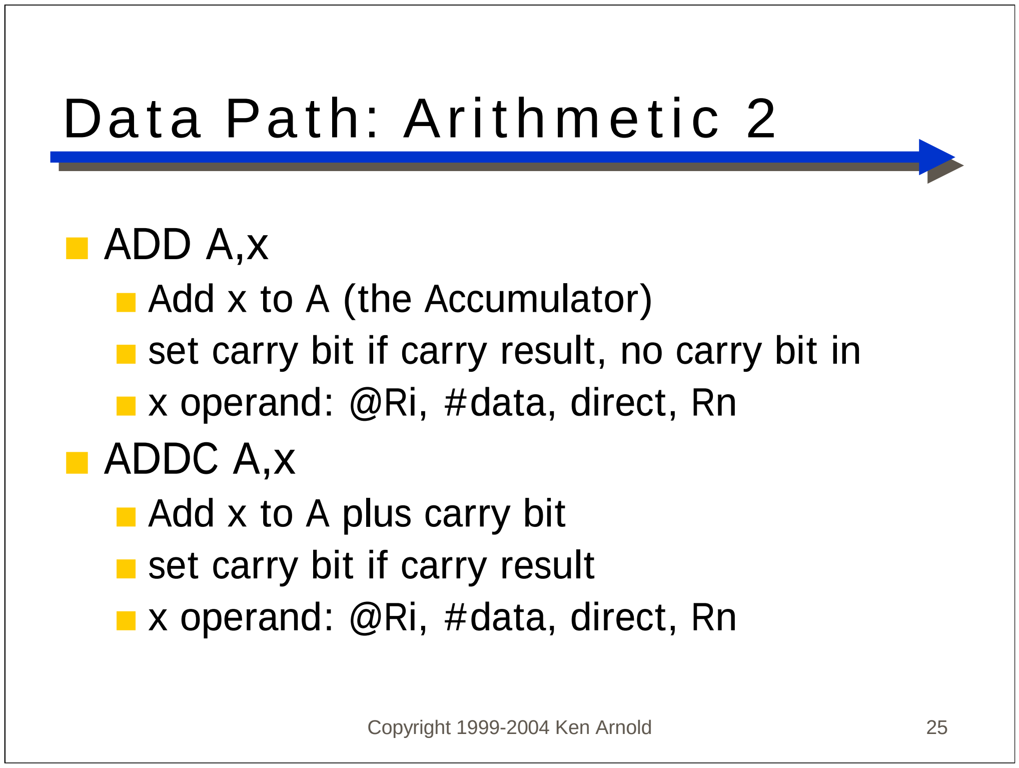
•For our multi-byte example, we first add 0FFh and 01, which results in 00 with CY=1. Then we use ADDC to add 1Fh and 02. 1Fh+2 = 21h then add in the carry flag for the final answer of 22h.

•Due to careful selection of the problem, it could be signed or unsigned. We really can’t tell. Why? Because the sign bit is the most significant bit of the multi-byte value. So, if our example is signed then it must be a positive value because the most significant bit of the most significant bytes (1Fh and 02) are zero.



Data Path: Arithmetic 1

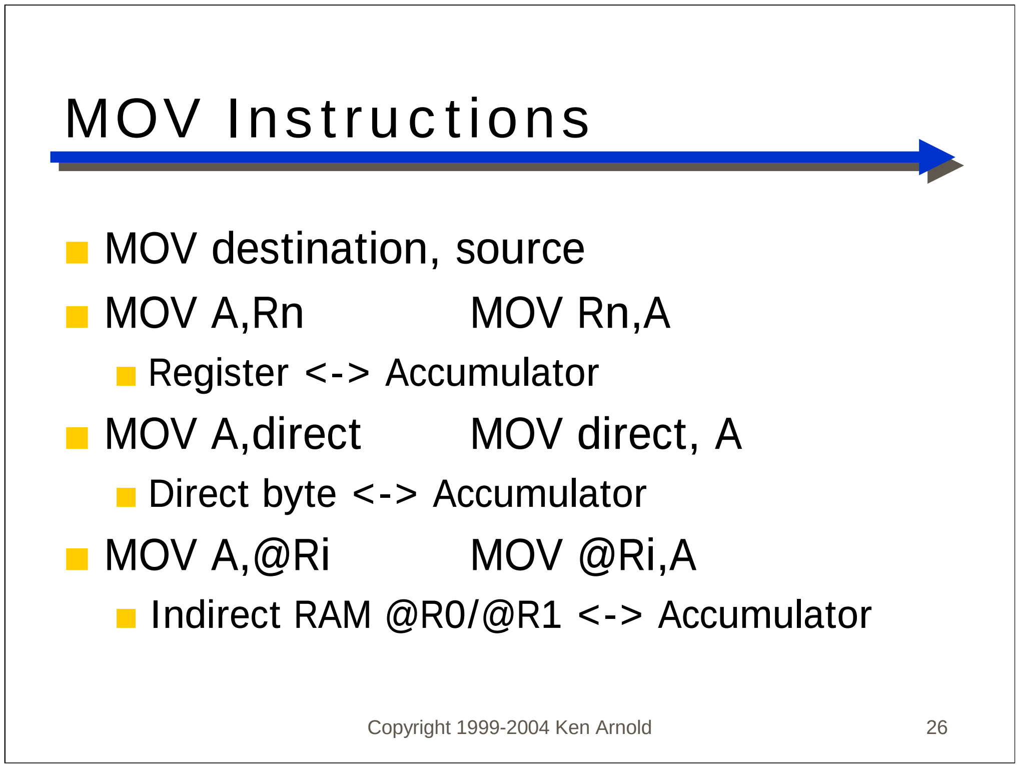
- INC/DEC
  - Increment or decrement
  - valid operands: A Rn @Ri direct
- INC dptr
  - Increments 16 bit dptr register
  - But there's no DEC dptr instruction!!



Data Path: Arithmetic 2

- ADD A,x
  - Add x to A (the Accumulator)
  - set carry bit if carry result, no carry bit in
  - x operand: @Ri, #data, direct, Rn
- ADDC A,x
  - Add x to A plus carry bit
  - set carry bit if carry result
  - x operand: @Ri, #data, direct, Rn

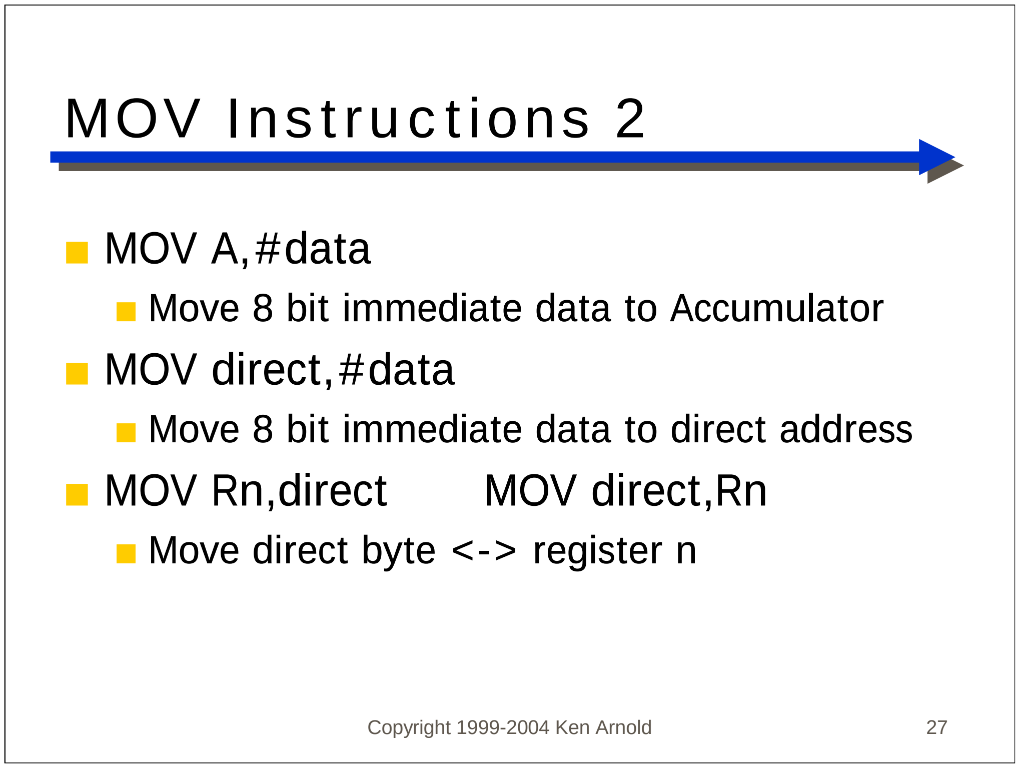
25



MOV Instructions

- MOV destination, source
- MOV A,Rn MOV Rn,A
  - Register <-> Accumulator
- MOV A,direct MOV direct, A
  - Direct byte <-> Accumulator
- MOV A,@Ri MOV @Ri,A
  - Indirect RAM @R0/@R1 <-> Accumulator

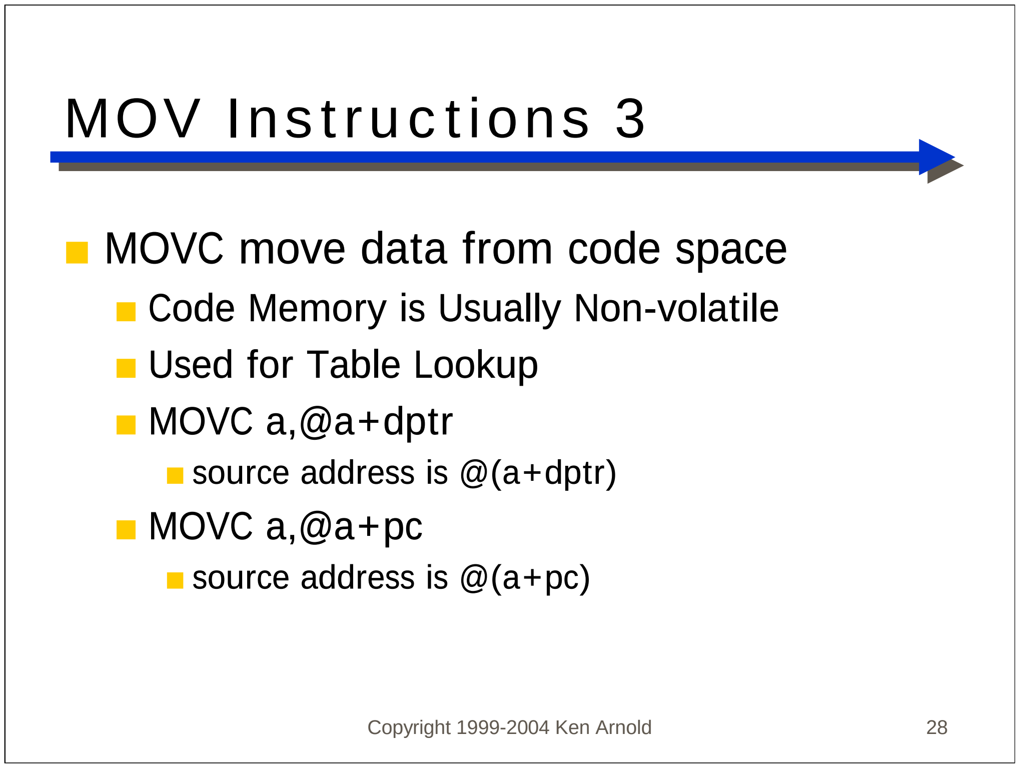
Copyright 1999-2004 Ken Arnold 26



MOV Instructions 2

- MOV A,#data
  - Move 8 bit immediate data to Accumulator
- MOV direct,#data
  - Move 8 bit immediate data to direct address
- MOV Rn,direct   MOV direct,Rn
  - Move direct byte <-> register n

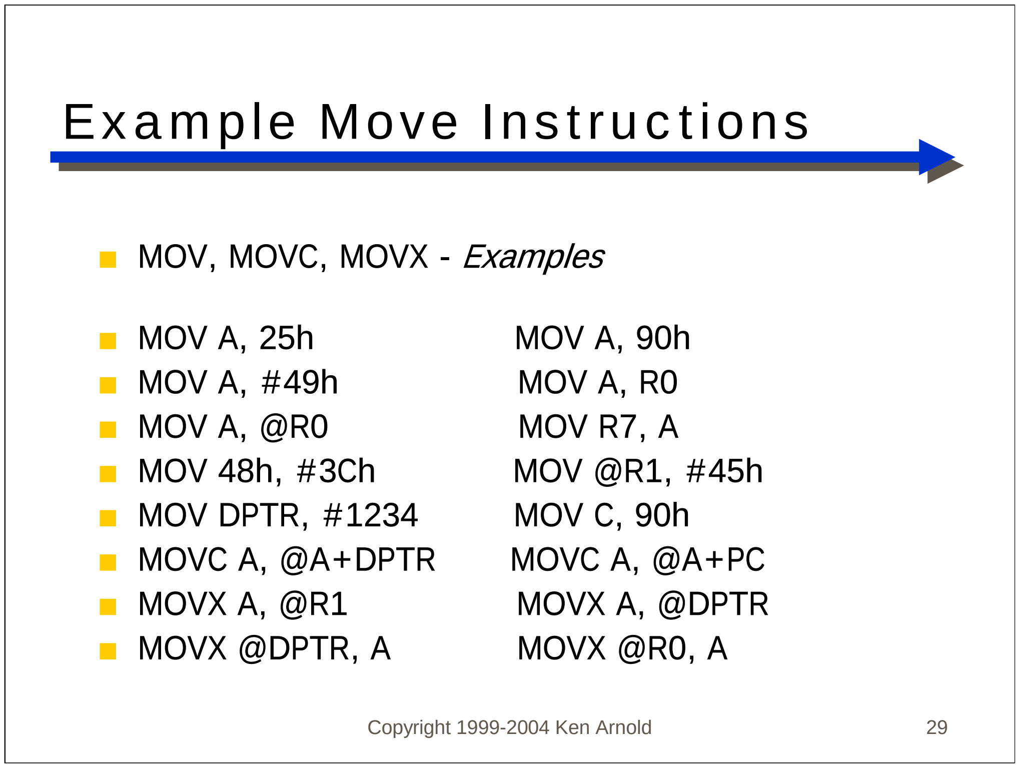
27



MOV Instructions 3

- MOVC move data from code space
  - Code Memory is Usually Non-volatile
  - Used for Table Lookup
  - MOVC a,@a+dptr
    - source address is @(a+dptr)
  - MOVC a,@a+pc
    - source address is @(a+pc)

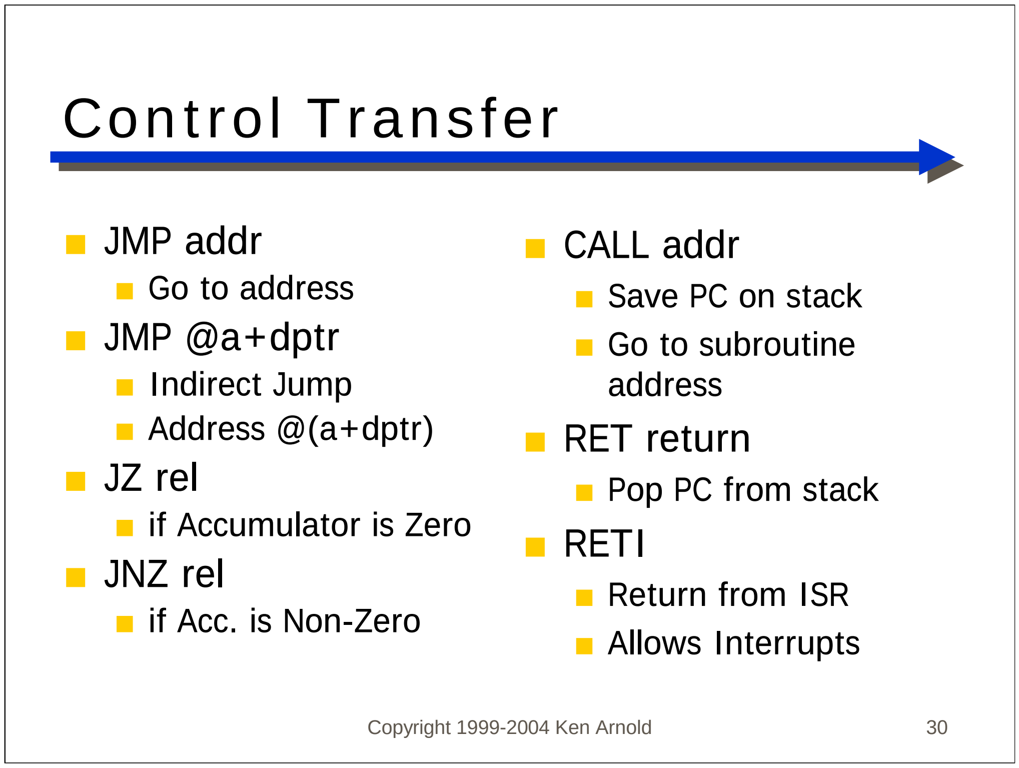
Copyright 1999-2004 Ken Arnold 28



Example Move Instructions

- MOV, MOVC, MOVX - *Examples*

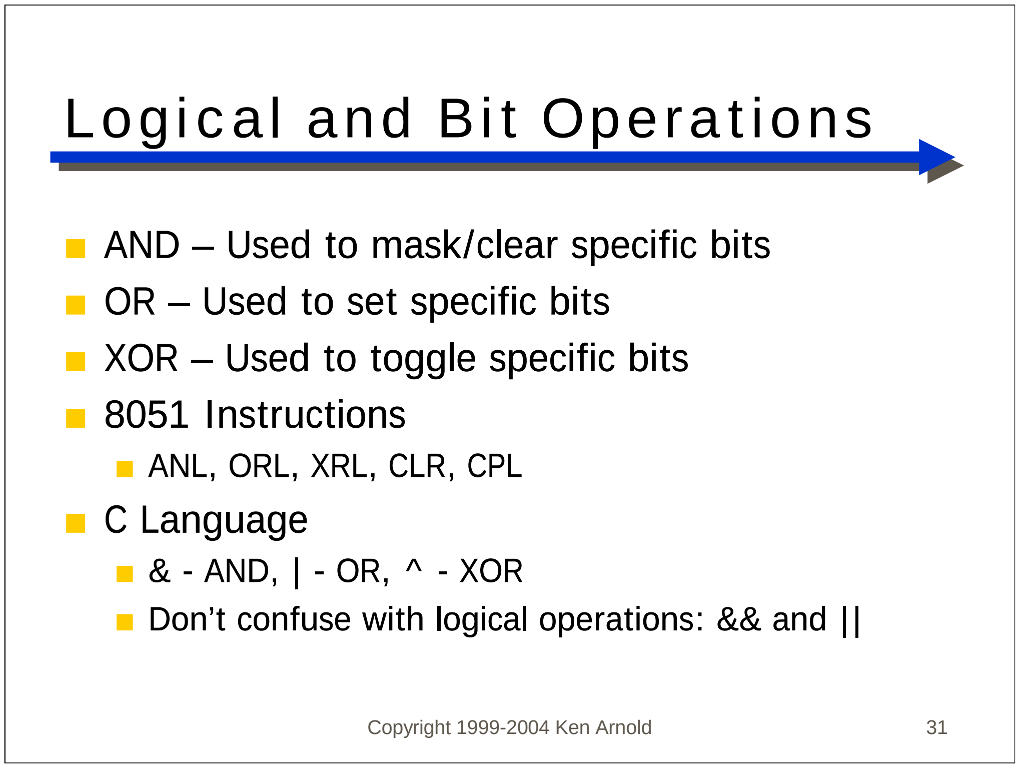
```
MOV A, 25h             MOV A, 90h
MOV A, #49h            MOV A, R0
MOV A, @R0             MOV R7, A
MOV 48h, #3Ch          MOV @R1, #45h
MOV DPTR, #1234        MOV C, 90h
MOVC A, @A+DPTR        MOVC A, @A+PC
MOVX A, @R1            MOVX A, @DPTR
MOVX @DPTR, A          MOVX @R0, A
```



Control Transfer

- JMP addr
  - Go to address
- JMP @a+dptr
  - Indirect Jump
  - Address @(a+dptr)
- JZ rel
  - if Accumulator is Zero
- JNZ rel
  - if Acc. is Non-Zero
- CALL addr
  - Save PC on stack
  - Go to subroutine address
- RET return
  - Pop PC from stack
- RETI
  - Return from ISR
  - Allows Interrupts

30



Logical and Bit Operations

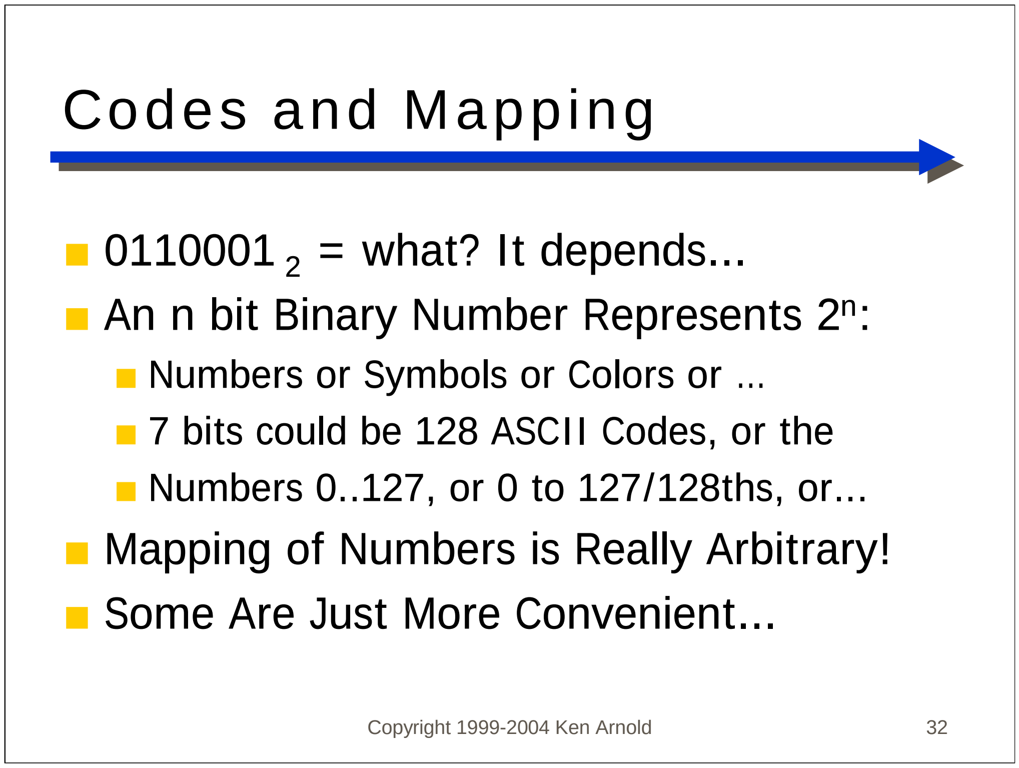
- AND – Used to mask/clear specific bits
- OR – Used to set specific bits
- XOR – Used to toggle specific bits
- 8051 Instructions
  - ANL, ORL, XRL, CLR, CPL
- C Language
  - & - AND, | - OR, ^ - XOR
  - Don’t confuse with logical operations: && and ||

31

•The use of logical operations is one of the basic foundations of any program. The 8051 supports the use of the standard suite of operations: AND, OR, XOR.

•With the 8051, the logical operations can operate on a byte or on a single bit.

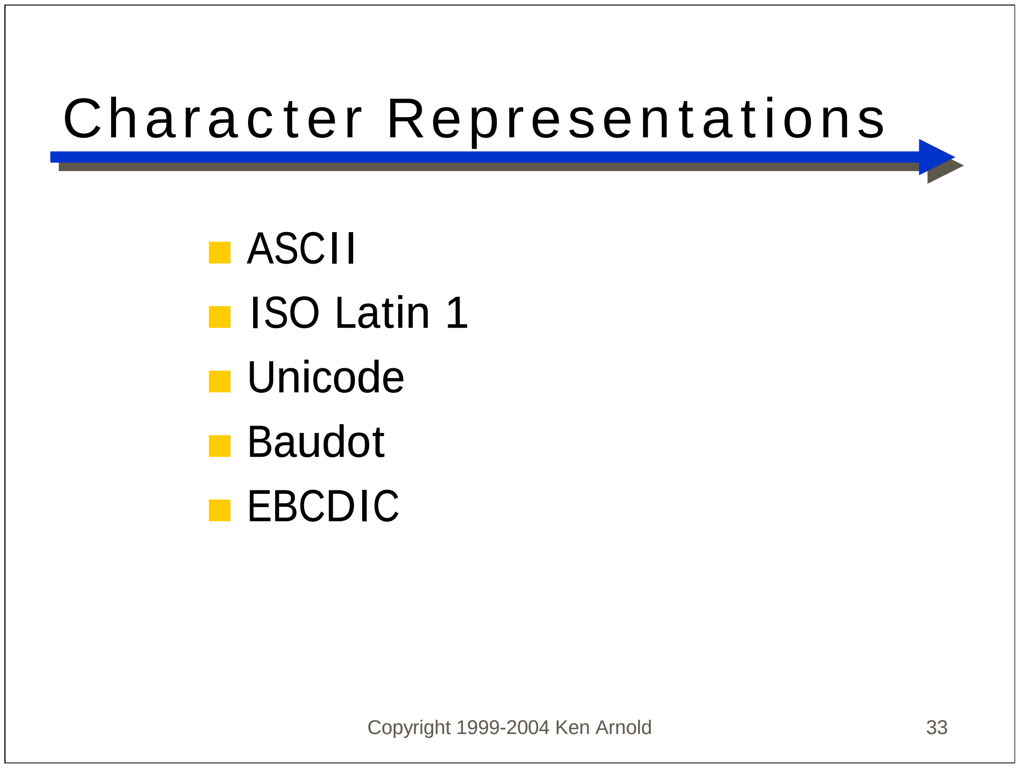
•The C language also supports these logical operations. It is important to note the difference between the logical operators and the Boolean operators.



Codes and Mapping

- 0110001_2 = what? It depends...
- An n bit Binary Number Represents 2^n:
  - Numbers or Symbols or Colors or ...
  - 7 bits could be 128 ASCII Codes, or the
  - Numbers 0..127, or 0 to 127/128ths, or...
- Mapping of Numbers is Really Arbitrary!
- Some Are Just More Convenient...

Copyright 1999-2004 Ken Arnold 32



Character Representations

- ASCII
- ISO Latin 1
- Unicode
- Baudot
- EBCDIC

•If you want to display information to the user, you must map the raw data to characters which the user can easily understand.

•There are many different standards for representing characters. The most commonly used are ASCII and, in recent years, Unicode.

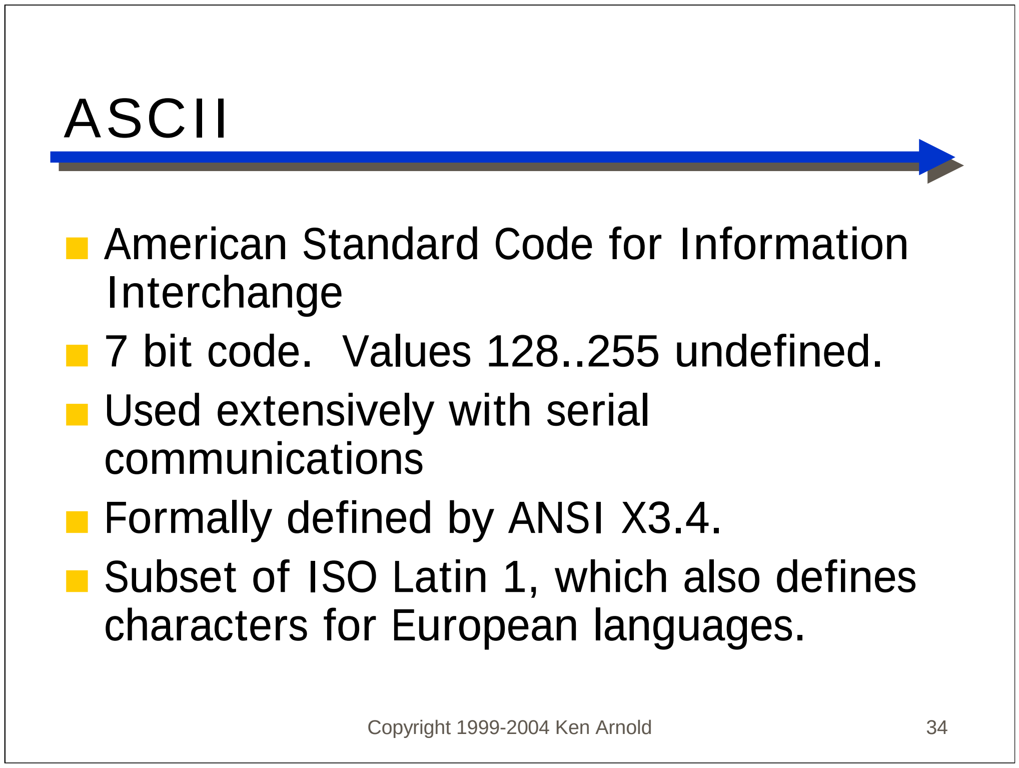
•ASCII is a seven-bit code, the most significant bit is unused. However, 128 values do not suffice to represent all characters in all languages in the word. ISO Latin 1 came along to add support for European languages, but it still couldn't address Far Eastern languages, for example. This is the reason for Unicode. Unicode supports many "pages" of mappings. ASCII is one of these pages.

•Baudot and EBCDIC are mentioned for historical reasons.

•Addition resources are:

> •ASCII information: http://dir.yahoo.com/Computers_and_Internet/Data_Formats/ASCII/
>
> •Unicode: http://www.unicode.org/

•ASCII is a code for representing English characters as numbers, with each letter assigned a number from 0 to 127. For example, the ASCII code for uppercase M is 77. We typically use ASCII codes to represent text, which makes it possible to display data to users.



ASCII

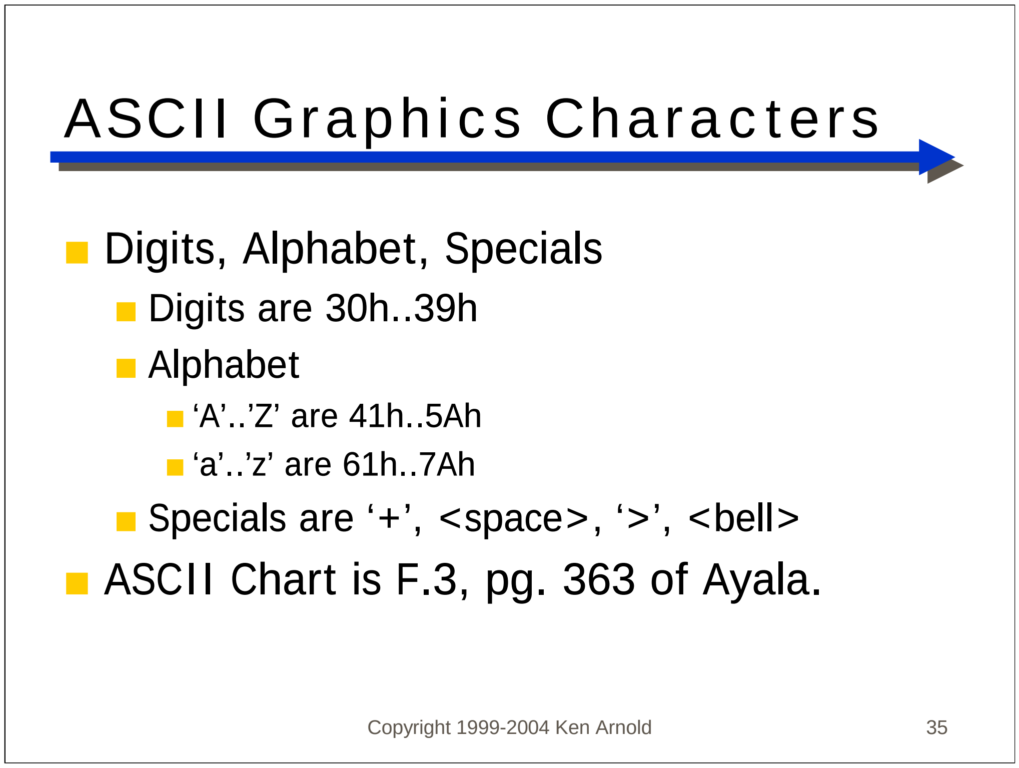
- American Standard Code for Information Interchange
- 7 bit code. Values 128..255 undefined.
- Used extensively with serial communications
- Formally defined by ANSI X3.4.
- Subset of ISO Latin 1, which also defines characters for European languages.

•ASCII codes are widely used to transmit data via serial communications links. Most computer devices have some sort of serial port using common standards like RS232 which allow easy interconnection of devices. The ASCII code provides a STANDARD so that different devices recognize the particular characters which are being sent. This greatly simplifies applications where text or control characters are being sent between devices.

•As an example, when your SDK talks to the PC it is connected to, it sends and receives ASCII characters over the serial port connection. Another fairly new way to add a text or graphic display to a microcontroller is to buy one of the new, low cost serial LCD modules which are becoming widely available.

•The creators of ASCII made it a 7 bit code. This means that value from 128 thru 255 (anything with a 1 in the most significant bit) are undefined. Different users define these to be any desired character, however, all users obviously have to understand what they mean. For example many word processing programs employ the undefined upper 128 characters as things like EOL, margin, or justification characters. ASCII characters are typically used and stored in that format in so called "text" documents. If you use your PC editor to create a basic text document, you will see that the characters are stored in the file in ASCII format.

•The size and scope of ASCII is formally defined in the American National Standards Institute (ANSI) standard X3.4.



ASCII Graphics Characters

- Digits, Alphabet, Specials
  - Digits are 30h..39h
  - Alphabet
    - ‘A’..’Z’ are 41h..5Ah
    - ‘a’..’z’ are 61h..7Ah
  - Specials are ‘+’, <space>, ‘>’, <bell>
- ASCII Chart is F.3, pg. 363 of Ayala.

Copyright 1999-2004 Ken Arnold 35

•The ASCII graphics characters are comprised of the Digits, Alphabet, and so called Specials.

•The digits start at 30h. That is, the character representing the digit 0 has an ASCII value of 30h. The rest of the digits 1 thru 9 are successively higher in the table. Converting between the numbers 0..9 and the characters ‘0’..’9’ is very easy. Just add 30h to the number and you get the character, subtract 30h from the character and you get the number.

•NOTE that although storing characters in this format is not economical, it is at least very portable !!!!

•The alphabet is represented in rows 4, 5, 6, and 7 of the ASCII chart. As an aside, note that the Latin Alphabet characters can be represented by a 5 bit character subset. This was often used in the past to adapt the code to older display types such as teleprinters which only displayed upper case type.

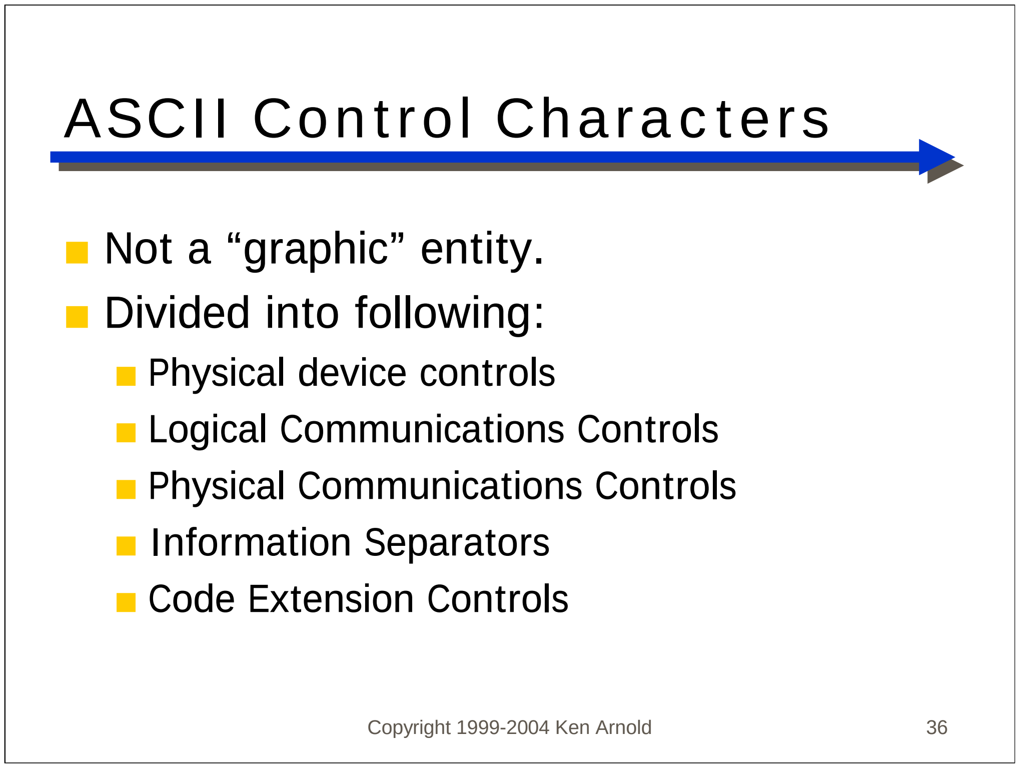
•How can we convert between lower and upper case ASCII values?

•The specials are mostly the other graphic characters found in lines 2 through 7. These characters include the SPACE and DEL characters which don't really print anything on the screen.

•Some of the special characters have been designated for international use. For example, the Swedish Alphabet has 29 characters. The international use characters, 5Eh, 60h, 7Eh can have other characters substituted as required by the local language. This means that you should avoid using them if your program has any chance of being used in these countries!

•The characters at 23h and 24h have been reserved for international currency symbols. This means, for example, that in England the # character may be replaced by the pound sterling symbol, £.

•ASCII collating - Rather than belabor the point, ASCII does not make a particularly good character set for collating (sorting) purposes. Of course if the sort is based on the alphabetic characters alone, it works okay but if names are separated by spaces, disaster strikes.



ASCII Control Characters

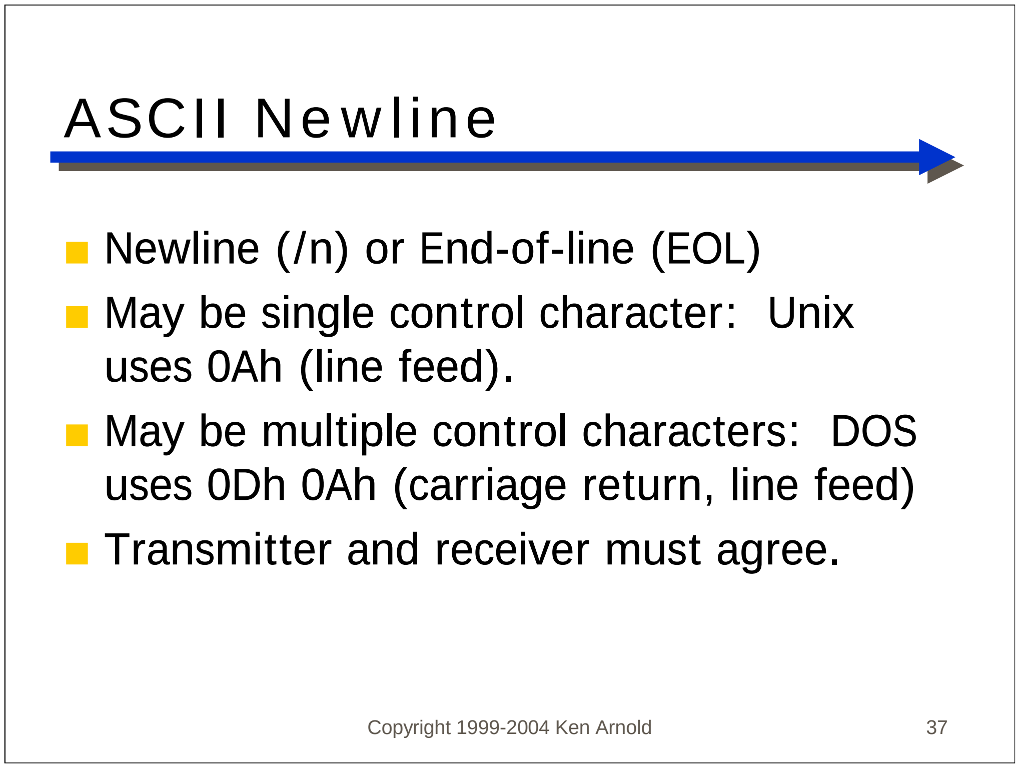
- Not a "graphic" entity.
- Divided into following:
  - Physical device controls
  - Logical Communications Controls
  - Physical Communications Controls
  - Information Separators
  - Code Extension Controls

36

•The ASCII control characters - All the previous characters have one thing in common - they represent some kind of graphic entity. I.E. They cause a specific character or symbol to appear on a display device. In contrast, the characters in ROWS 1 and 2, together with the DEL character at the end of row 7, do not generate any graphic characters.

•They are known as the control characters and can be roughly divided into the following characters: Physical Device Controls, Logical Communications Controls, Physical Communications Controls, Information Separators, and Code Extension Controls.

•There are considerably more details to the use of control characters which go beyond the course of this class. Many good references on serial communications deal with these ASCII control characters in much greater detail. If you are working with modems and their various protocols, control characters play a big role. Refer to a good reference for more details.



ASCII Newline

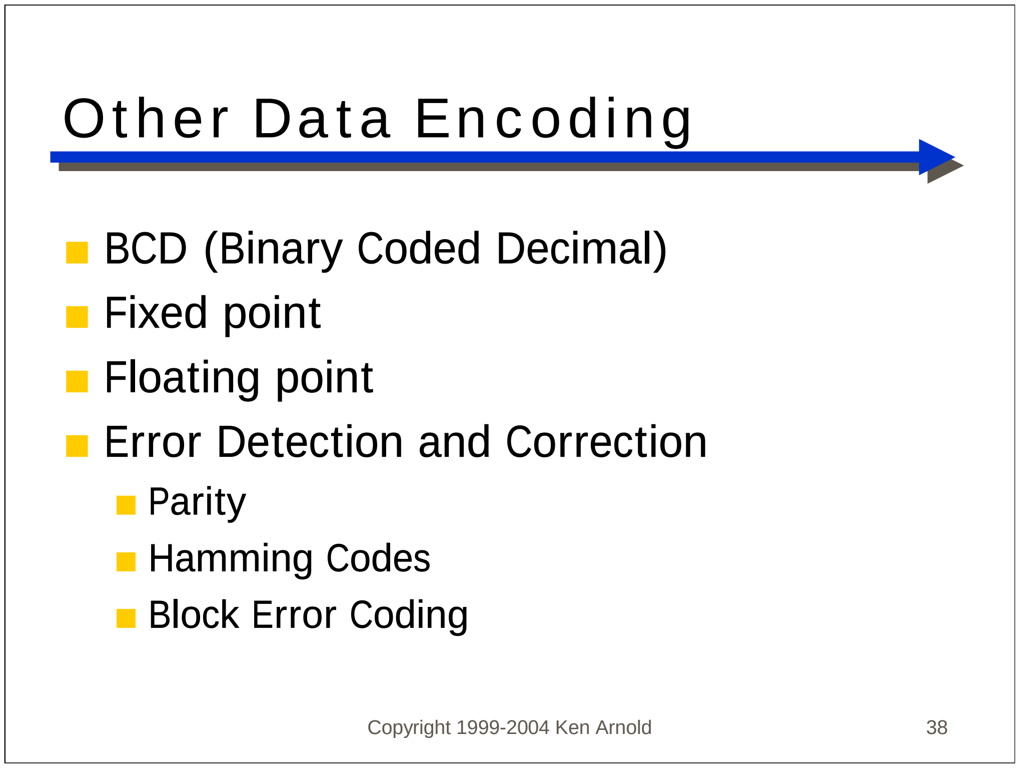
- Newline (/n) or End-of-line (EOL)
- May be single control character: Unix uses 0Ah (line feed).
- May be multiple control characters: DOS uses 0Dh 0Ah (carriage return, line feed)
- Transmitter and receiver must agree.

•One other ASCII issue: the ubiquitous NEWLINE character. This character (which may be a combination of characters), causes no end of problems when talking to other serial display devices.

•Character 0Dh is the character for Carriage Return (CR). Sometimes this character returns to the beginning of the next line, sometimes to the beginning of the CURRENT line.

•Character 0Ah is the character for Line Feed (LF). This character sometimes returns to the beginning of the next line, and sometimes to the CURRENT column on the next line. This LF character is also known as NEWLINE but only when sender and receiver agree! This definition is unfortunately vague in the ASCII spec itself; thus the problem. This also leads to the unfortunate problem that pure ASCII text files are not portable over different systems. UNIX, for example, always uses the New Line Convention, while PC DOS uses CR/LF. To further compound the problem Apple and Commodore use CR!

•So, when using these characters, know your display device.

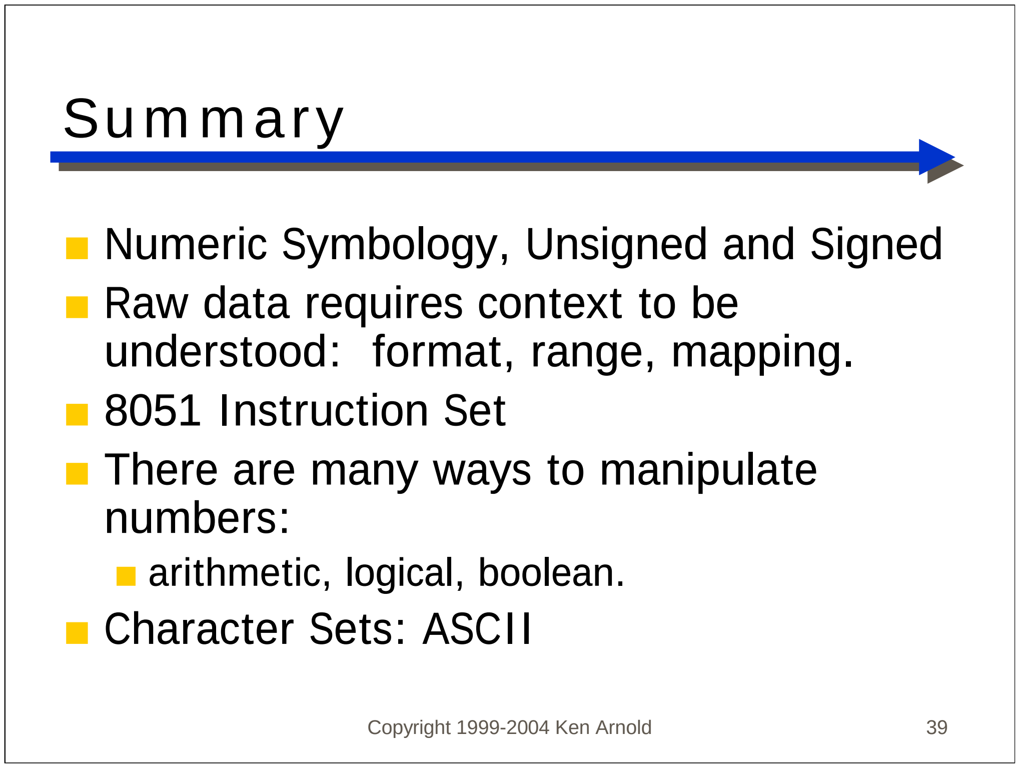


Other Data Encoding

- BCD (Binary Coded Decimal)
- Fixed point
- Floating point
- Error Detection and Correction
  - Parity
  - Hamming Codes
  - Block Error Coding

38

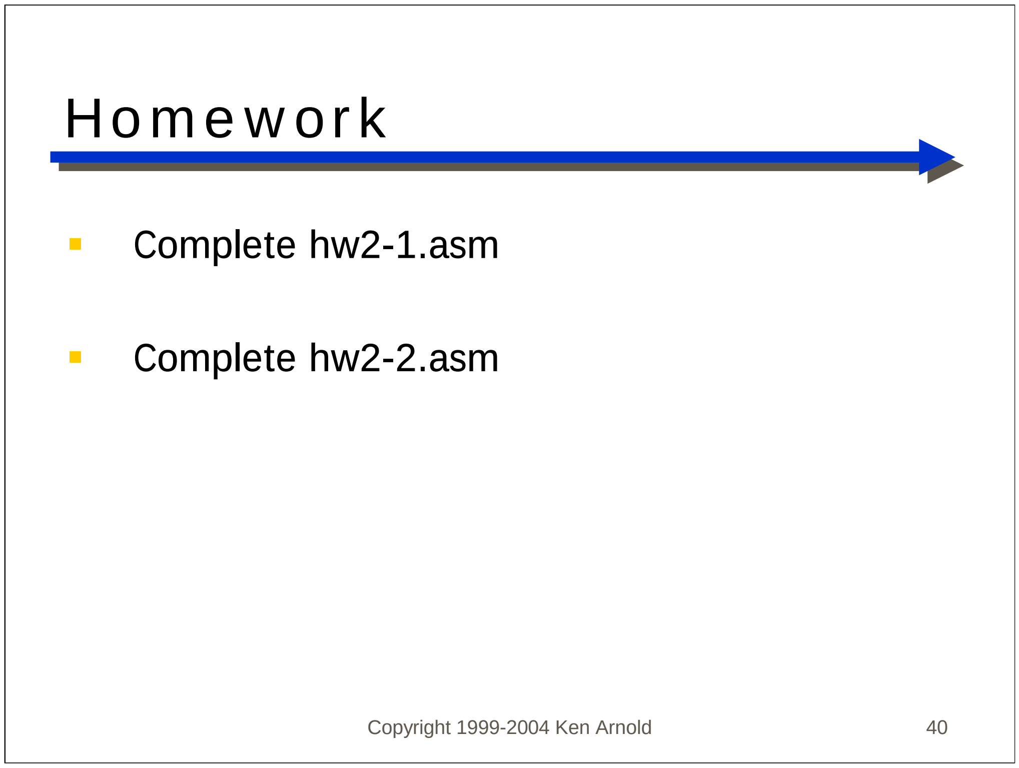
- Decimal numbers are rather ubiquitous in everyday life, and to allow computers to more easily deal with decimals, another code has evolved name Binary Coded Decimal or BCD. Basically BCD is the radix-2 coding of the binary data 0..9. So, the packed BCD coding for 24_{10} would be $(0010\ 0100)_2$. The 8051 has specific instructions to deal with BCD: DA (Decimal Adjust) and the AC flag of the PSW.
- Fixed point is a mechanism to specify that some number of bits are to the right of the decimal point. You can thing of the signed and unsigned numbers we've been talking about as fixed point numbers with 0 bits to the right of the decimal point…that is, they are integers. We'll talk more about fixed point numbers later.
- Floating point encoding can be used to define values that approximate real-numbers, such as π, 1/3, etc. There is no support on the 8051 for floating point math, consequently it is very slow. Note that the C compiler will insert a floating point library for you if you try to use floating point variables in C.
- There are many error correction and detection codes available. The most commonly used in serial communications is "parity". Another common encoding scheme is termed "Forward Error Correction" or FEC. Unfortunately, FEC is a huge topic that could easily consume an entire semester.



Summary

- Numeric Symbology, Unsigned and Signed
- Raw data requires context to be understood: format, range, mapping.
- 8051 Instruction Set
- There are many ways to manipulate numbers:
  - arithmetic, logical, boolean.
- Character Sets: ASCII

39



Homework

- Complete hw2-1.asm
- Complete hw2-2.asm

40