1. Examples of Subtraction

```
A-B              A+(TC(B))

00110110         00110110
00110100-        11001100+
00000010        100000010

borrow = 0       end-carry = 1
```

2. Examples of Subtraction

```
 A-B              A+(TC(B))

 00001000         00001000
1 01111111-       10000001+
 10001001         10001001

 borrow = 1       end-carry = 0
```

C-Flag

After subtraction, the C-flag records whether a borrow occurred and not a carry!

Overflow

If the result of an addition or subtraction of signed numbers results in an answer that is outside the range of the signed number system, then overflow occurred.

```
e.g.:
    01110000         112
    01000000+         64+
    10110000         176
```

If we interpret 10110000 as a signed number then the result is -80 -> wrong answer.

Limitation of the Number System

- One solution:
  - Increase the number of bits
  - But, it will always be possible to cause overflow by choosing large enough values!
- What do we do?
  - The CPU provides a CCR bit called the oVerflow flag (V).
  - Set it the CPU detects that overflow has occurred during arithmetic operations.

The oVerflow flag (V)

```
                    Signed   Unsigned
1   01110000          112       112
    10110000+         -80+      176+
    00100000           32       288

    Carry=1, oVerflow=0
```

We have two valid results!

```
2
                   Signed   Unsigned
    10110000        -80       176
    10110000+       -80+      176+
    01100000         96        96

    Carry=1, oVerflow=1
```

Unsigned result:

`01100000 C=1`

⇒ Carry acts as an extra bit in the result

⇒ (**96 + 256 = 352**)

Signed result:

`01100000 V=1`

⇒ Result out of range.

1. Overflow Identification Rule

Addition

r = a + b

V = 1 if MSB(a) = MSB(b) and
MSB(r) 🕒 MSB(a)

i.e.: overflow occurs for addition if the operands are the same sign and the result is of a different sign.

2. Overflow Identification Rule

Subtraction

r = a - b

V = 1 if MSB(a) 🕒 MSB(b) and
MSB(r) 🕒 MSB(a)

i.e.: overflow can only occur if the operands are of different signs and if the sign of the result is different from the sign of the first operand.

Overflow

The oVerflow flag provides us with a means to effectively deal with binary arithmetic using 2's complement negative numbers. If we are not interpreting the number as negative then we simply ignore the V flag.

Other Coding Schemes

- Decimal Codes
  - BCD (Binary coded decimal)
    - Represent each decimal digit directly in binary
    - Need 4 bits to represent 10 unique states
    - Use 4 bits, let the last 6 states go unused
  - E.g. $(325)_{10}$ in Binary = $(101000101)_2$
    $(325)_{10}$ in BCD = $(0011\ 0010\ 0101)_2$

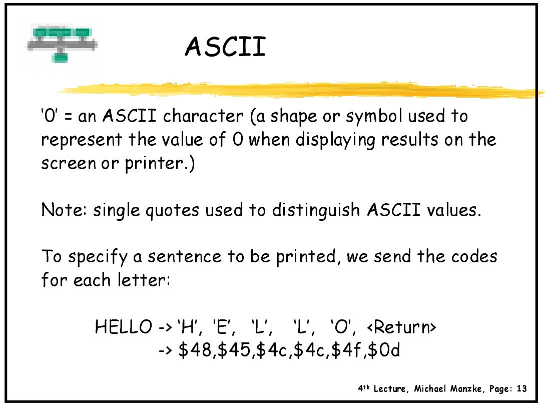
Other Coding Schemes

ASCII

Used to encode alphanumeric and other Characters are associated with text based I/O.

Each character symbol is coded as a unique binary value. The **ASCII** (**A**merican **S**tandard **C**ode **I**nformation **I**nterchange) uses 1 byte per symbol.

Note: the character '0' is not the same as the value 0.



ASCII

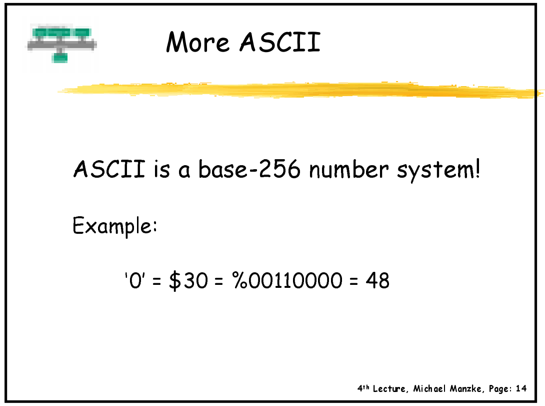
‘0’ = an ASCII character (a shape or symbol used to represent the value of 0 when displaying results on the screen or printer.)

Note: single quotes used to distinguish ASCII values.

To specify a sentence to be printed, we send the codes for each letter:

HELLO -> ‘H’, ‘E’, ‘L’, ‘L’, ‘O’, <Return>
-> $48,$45,$4c,$4c,$4f,$0d

4th Lecture, Michael Manzke, Page: 13



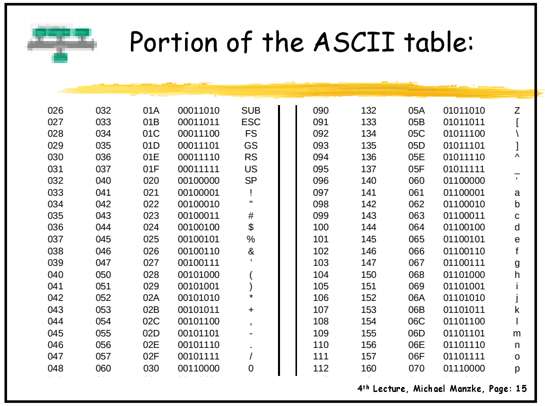
More ASCII

ASCII is a base-256 number system!

Example:

‘0’ = $30 = %00110000 = 48

4th Lecture, Michael Manzke, Page: 14



Portion of the ASCII table:

026	032	01A	00011010	SUB	090	132	05A	01011010	Z
027	033	01B	00011011	ESC	091	133	05B	01011011	[
028	034	01C	00011100	FS	092	134	05C	01011100	\
029	035	01D	00011101	GS	093	135	05D	01011101	]
030	036	01E	00011110	RS	094	136	05E	01011110	^
031	037	01F	00011111	US	095	137	05F	01011111	_
032	040	020	00100000	SP	096	140	060	01100000	‘
033	041	021	00100001	!	097	141	061	01100001	a
034	042	022	00100010	"	098	142	062	01100010	b
035	043	023	00100011	#	099	143	063	01100011	c
036	044	024	00100100	$	100	144	064	01100100	d
037	045	025	00100101	%	101	145	065	01100101	e
038	046	026	00100110	&	102	146	066	01100110	f
039	047	027	00100111	’	103	147	067	01100111	g
040	050	028	00101000	(	104	150	068	01101000	h
041	051	029	00101001	)	105	151	069	01101001	i
042	052	02A	00101010	*	106	152	06A	01101010	j
043	053	02B	00101011	+	107	153	06B	01101011	k
044	054	02C	00101100	,	108	154	06C	01101100	l
045	055	02D	00101101	-	109	155	06D	01101101	m
046	056	02E	00101110	.	110	156	06E	01101110	n
047	057	02F	00101111	/	111	157	06F	01101111	o
048	060	030	00110000	0	112	160	070	01110000	p

4th Lecture, Michael Manzke, Page: 15