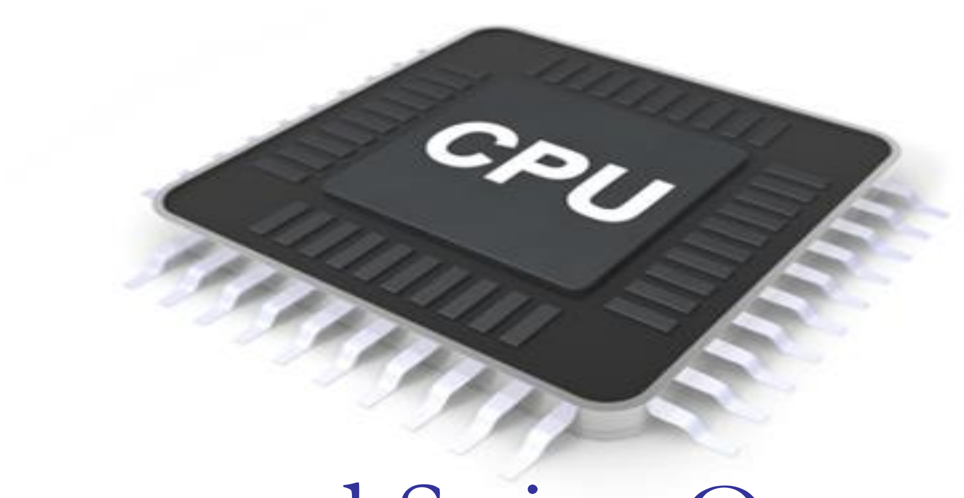


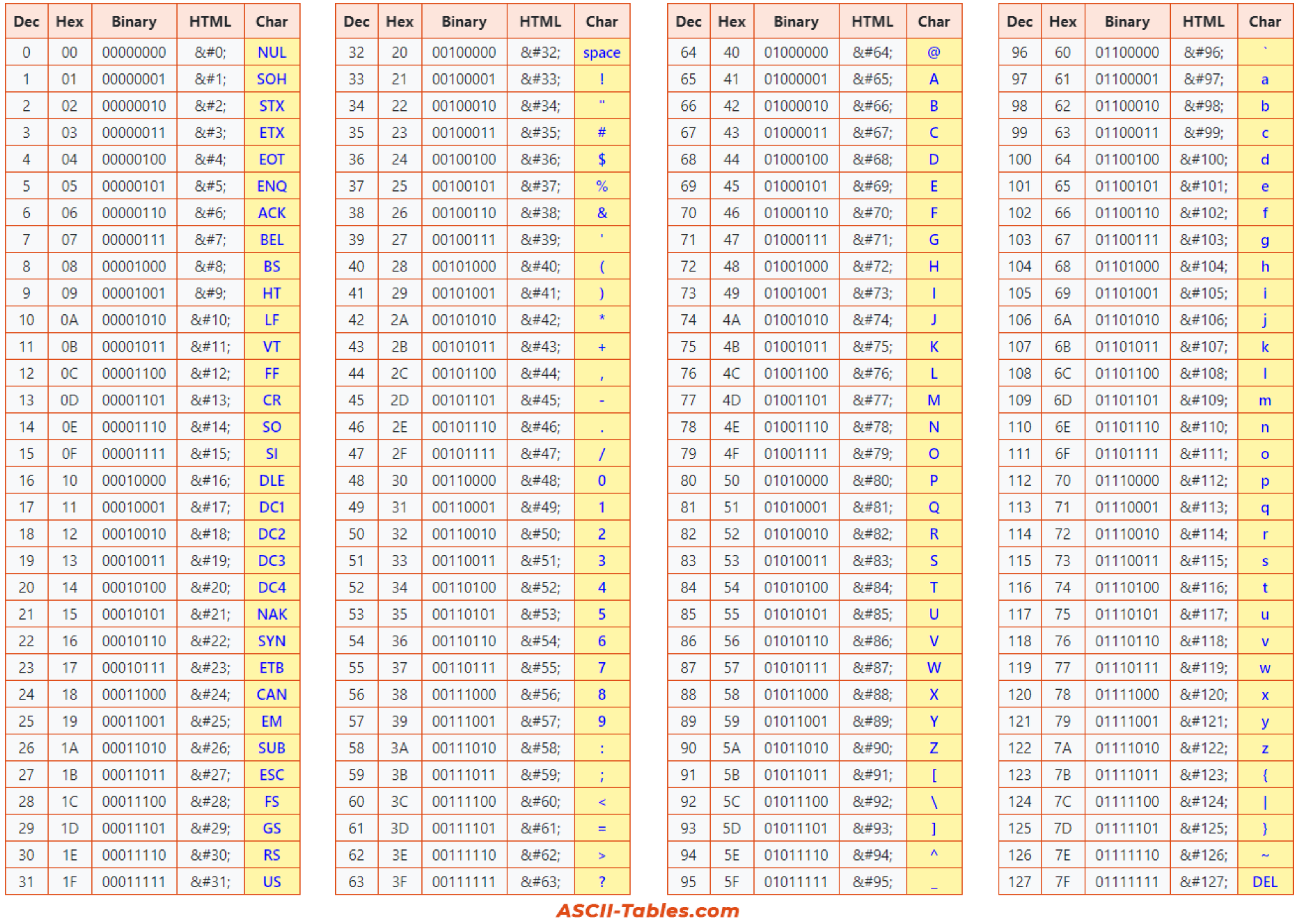
Array and String Operations

ENEE 3582

Microp

Strings

- ASCII arrays of bytes
  - American Standard Code for Information Interchange



Dec	Hex	Binary	HTML	Char	Dec	Hex	Binary	HTML	Char	Dec	Hex	Binary	HTML	Char	Dec	Hex	Binary	HTML	Char
0	00	00000000	&#0;	NUL	32	20	00100000	&#32;	space	64	40	01000000	&#64;	@	96	60	01100000	&#96;	`
1	01	00000001	&#1;	SOH	33	21	00100001	&#33;	!	65	41	01000001	&#65;	A	97	61	01100001	&#97;	a
2	02	00000010	&#2;	STX	34	22	00100010	&#34;	"	66	42	01000010	&#66;	B	98	62	01100010	&#98;	b
3	03	00000011	&#3;	ETX	35	23	00100011	&#35;	#	67	43	01000011	&#67;	C	99	63	01100011	&#99;	c
4	04	00000100	&#4;	EOT	36	24	00100100	&#36;	$	68	44	01000100	&#68;	D	100	64	01100100	&#100;	d
5	05	00000101	&#5;	ENQ	37	25	00100101	&#37;	%	69	45	01000101	&#69;	E	101	65	01100101	&#101;	e
6	06	00000110	&#6;	ACK	38	26	00100110	&#38;	&	70	46	01000110	&#70;	F	102	66	01100110	&#102;	f
7	07	00000111	&#7;	BEL	39	27	00100111	&#39;	'	71	47	01000111	&#71;	G	103	67	01100111	&#103;	g
8	08	00001000	&#8;	BS	40	28	00101000	&#40;	(	72	48	01001000	&#72;	H	104	68	01101000	&#104;	h
9	09	00001001	&#9;	HT	41	29	00101001	&#41;	)	73	49	01001001	&#73;	I	105	69	01101001	&#105;	i
10	0A	00001010	&#10;	LF	42	2A	00101010	&#42;	*	74	4A	01001010	&#74;	J	106	6A	01101010	&#106;	j
11	0B	00001011	&#11;	VT	43	2B	00101011	&#43;	+	75	4B	01001011	&#75;	K	107	6B	01101011	&#107;	k
12	0C	00001100	&#12;	FF	44	2C	00101100	&#44;	,	76	4C	01001100	&#76;	L	108	6C	01101100	&#108;	l
13	0D	00001101	&#13;	CR	45	2D	00101101	&#45;	-	77	4D	01001101	&#77;	M	109	6D	01101101	&#109;	m
14	0E	00001110	&#14;	SO	46	2E	00101110	&#46;	.	78	4E	01001110	&#78;	N	110	6E	01101110	&#110;	n
15	0F	00001111	&#15;	SI	47	2F	00101111	&#47;	/	79	4F	01001111	&#79;	O	111	6F	01101111	&#111;	o
16	10	00010000	&#16;	DLE	48	30	00110000	&#48;	0	80	50	01010000	&#80;	P	112	70	01110000	&#112;	p
17	11	00010001	&#17;	DC1	49	31	00110001	&#49;	1	81	51	01010001	&#81;	Q	113	71	01110001	&#113;	q
18	12	00010010	&#18;	DC2	50	32	00110010	&#50;	2	82	52	01010010	&#82;	R	114	72	01110010	&#114;	r
19	13	00010011	&#19;	DC3	51	33	00110011	&#51;	3	83	53	01010011	&#83;	S	115	73	01110011	&#115;	s
20	14	00010100	&#20;	DC4	52	34	00110100	&#52;	4	84	54	01010100	&#84;	T	116	74	01110100	&#116;	t
21	15	00010101	&#21;	NAK	53	35	00110101	&#53;	5	85	55	01010101	&#85;	U	117	75	01110101	&#117;	u
22	16	00010110	&#22;	SYN	54	36	00110110	&#54;	6	86	56	01010110	&#86;	V	118	76	01110110	&#118;	v
23	17	00010111	&#23;	ETB	55	37	00110111	&#55;	7	87	57	01010111	&#87;	W	119	77	01110111	&#119;	w
24	18	00011000	&#24;	CAN	56	38	00111000	&#56;	8	88	58	01011000	&#88;	X	120	78	01111000	&#120;	x
25	19	00011001	&#25;	EM	57	39	00111001	&#57;	9	89	59	01011001	&#89;	Y	121	79	01111001	&#121;	y
26	1A	00011010	&#26;	SUB	58	3A	00111010	&#58;	:	90	5A	01011010	&#90;	Z	122	7A	01111010	&#122;	z
27	1B	00011011	&#27;	ESC	59	3B	00111011	&#59;	;	91	5B	01011011	&#91;	[	123	7B	01111011	&#123;	{
28	1C	00011100	&#28;	FS	60	3C	00111100	&#60;	<	92	5C	01011100	&#92;	\	124	7C	01111100	&#124;	|
29	1D	00011101	&#29;	GS	61	3D	00111101	&#61;	=	93	5D	01011101	&#93;	]	125	7D	01111101	&#125;	}
30	1E	00011110	&#30;	RS	62	3E	00111110	&#62;	>	94	5E	01011110	&#94;	^	126	7E	01111110	&#126;	~
31	1F	00011111	&#31;	US	63	3F	00111111	&#63;	?	95	5F	01011111	&#95;	_	127	7F	01111111	&#127;	DEL

ASCII-Tables.com

String Copying: PM to DM

- String to copy from is in PM
- String to copy into is in DM
- If strings are null-terminated you don’t need to use length
  - String ends with 0

String Copy

Using string length:

```
        LDI ZH, HIGH(2*PMstr)
        LDI ZL, LOW(2*PMstr)

        LDI XH, HIGH(DMstr)
        LDI XL, LOW(DMstr)

        LDI R17, len
L1:     LPM R16, Z+
        ST  X+, r16

        DEC R17
        TST R17
        BRNE L1
```

Using null-terminated strings

```
        LDI ZH, HIGH(2*PMstr)
        LDI ZL, LOW(2*PMstr)

        LDI XH, HIGH(DMstr)
        LDI XL, LOW(DMstr)

L1:     LPM R16, Z+
        ST  X+, r16

        TST R16
        BRNE L1
```

String Copy

SMOV1 PMstr, DMstr, len

```
.MACRO SMOV1
         LDI ZH, HIGH(2*@0)
         LDI ZL, LOW(2*@0)

         LDI XH, HIGH(@1)
         LDI XL, LOW(@1)

         LDI R17, @2
SMOV_L1: LPM R16, Z+
         ST  X+, r16

         DEC R17
         TST R17
         BRNE SMOV_L1

.ENDM
```

SMOV2 PMstr, DMstr ;null-terminated

```
.MACRO SMOV2
         LDI ZH, HIGH(2*@0)
         LDI ZL, LOW(2*@0)

         LDI XH, HIGH(@1)
         LDI XL, LOW(@1)

SMOV_L1: LPM R16, Z+
         ST  X+, r16

         TST R16
         BRNE SMOV_L1

.ENDM
```

String SCAN

- Look for a character in the string
  - Character and string are in PM
  - Assume null-terminated string
  - Set R20 if found

```
          LDI ZH, HIGH(2*PMchar)
          LDI ZL, LOW(2*PMchar)
          LPM R16, Z

          LDI ZH, HIGH(2*PMstr)
          LDI ZL, LOW(2*PMstr)
          CLR R20
L1:       LPM R17, Z+
          CP R17, R16
          BREQ FND

          TST R17
          BRNE L1
          RJMP NOTFND
FND:      LDI R20, 1
NOTFND:
```

String SCAN

```
.MACRO  SSCAN                           ;sscan @0=char var, @1=string var, @2=found (reg)=0/1
        LDI ZH, HIGH(2*@0)
        LDI ZL, LOW(2*@0)
        LPM R16, Z

        LDI ZH, HIGH(2*@1)
        LDI ZL, LOW(2*@1)
        CLR @2
L1:     LPM R17, Z+
        CP R17, R16
        BREQ FND

        TST R17
        BRNE L1
        RJMP NOTFND
FND:    LDI @2, 1
NOTFND:
.ENDM
```

SCAN Returns Index of Found Character

```
.MACRO  SSCAN                       ;sscan @0=char var, @1=string var, @2=REG INDEX IF FOUND, -1 OW
        LDI ZH, HIGH(2*@0)
        LDI ZL, LOW(2*@0)
        LPM R16, Z

        LDI ZH, HIGH(2*@1)
        LDI ZL, LOW(2*@1)
        LDI @2, -1                  ;ASSUME NOT FOUND => INDEX = -1
        CLR R21                     ;Counter used for index
L1:     LPM R17, Z+
        CP R17, R16
        BREQ FND
        INC R21                     ;R21 = IX
        TST R17
        BRNE L1
        RJMP NOTFND
FND:    MOV @2, R21                 ;FOUND, INDEX
NOTFND:
.ENDM
```

String Compare

- Assume strings are the same size, null-terminated strings in PM
- Alphabetic ordering:
  - Uppercase has a smaller value than lowercase, ie “A” < “a”
  - Punctuation characters have smaller values than letters, ie “A” > “ “
  - Some special characters will be between “Z” and “a”
- When done use:
  - BREQ: check if strings are equal
  - BRLO: check if string1 < string 2
  - BRSH: check if string1 >= string 2

String Compare

```
              CLR R16                       ;IX
L1:           LDI ZH, HIGH (PMstr1*2)
              LDI ZL, LOW  (PMstr1*2)
              ADD ZL, R16
              LPM R17, Z
              CPI R17, 0
              BREQ Done

              LDI ZH, HIGH (PMstr2*2)
              LDI ZL, LOW  (PMstr2*2)
              ADD ZL, R16
              LPM R18, Z

              INC R16
              CP R17, R18
              BREQ L1
Done:
```

String Compare

```
.MACRO  SCOMP                                   ;@0=STR1 VAR, @2=STR2 VAR
               CLR R16                          ;IX
L1:            LDI ZH, HIGH (@0*2)
               LDI ZL, LOW  (@0*2)
               ADD ZL, R16
               LPM R17, Z
               CPI R17, 0
               BREQ Done

               LDI ZH, HIGH (@1*2)
               LDI ZL, LOW  (@1*2)
               ADD ZL, R16
               LPM R18, Z

               INC R16
               CP R17, R18
               BREQ L1
Done:
.ENDM
```

List: Array of Strings

- Strings could be fixed length
- For variable length strings
  - Use null to determine end of string
  - Use 2 nulls to determine end of list
- Applications:
  - Scan for a character in the entire list
  - Scan for a character in each string
  - Copy a 1 string from the list into another string
  - Search for string in the list

Scan The List

- List contains variable size null-terminated strings, find how many characters strings are there in the list. The list is terminated with two nulls.
- Idea:
  - Scan for the 00
  - Count nulls => array count
  - SET carry if first 0 is encountered.

```
        LDI ZH, HIGH (PMList*2)
        LDI ZL, LOW  (PMList*2)
L1:     LPM R17, Z+
        TST R17
        BRNE next
        BRCS Done
        INC R16             ;string count
        SEC
        RJMP L1
next:   CLC
        RJMP L1
Done:
```

MACRO Scan The List

```
.MACRO  LstScan                 ;@0=list, @1=number of strings
        LDI ZH, HIGH (@0*2)
        LDI ZL, LOW  (@0*2)
L1:     LPM R17, Z+
        TST R17
        BRNE next
        BRCS Done
        INC @1
        SEC
        RJMP L1
next:   CLC
        RJMP L1
Done:
.ENDM
```

Scan Each String in a List

- List of 20 zipcodes. Find how many occurrences of “7” are there in the list.
- Assume 5 digits for each zipcode
  - List has fixed length strings

```
                LDI ZH, HIGH (CHAR*2)
                LDI ZL, LOW  (CHAR*2)
                LPM R19, Z

                LDI ZH, HIGH (PMList*2)
                LDI ZL, LOW  (PMList*2)
                LDI R16, 20
outer:          LDI R17, 5
inner:          LPM R18, Z+
                CP R18, R19
                BRNE nextchar
                INC R20          ;count
next:           DEC R17
                TST R17
                BRNE inner
                DEC R16
                TST R16
                BRNE outer

Done:
```

Copy a String from a List

- List1 contains 20 names, each is 16 characters long. Copy the 10th name into a variable in the DM.

```
        LDI ZH, HIGH (PMList*2)
        LDI ZL, LOW  (PMList*2)

        LDI R16, 10-1
        LDI R17, 16
        MUL R17, R16

        ADD ZL, R0
        ADC ZH, R1

        LDI XH, HIGH(DMstr)
        LDI XL, LOW(DMstr)
L1:     LPM R16, Z+
        ST X+, R16
        DEC R17
        TST R17
        BRNE L1
```

Copy Any String from a List: MACRO

```
.MACRO  LCOPY
        LDI ZH, HIGH (@0*2)   ;@0=LIST VAR
        LDI ZL, LOW  (@0*2)

        LDI XH, HIGH(@1)      ;@1=DM STR VAR
        LDI XL, LOW(@1)

        LDI R16, @2-1         ;@2 = INDEX OF STRING IN THE LIST
        LDI R17, @3           ;@3 = LENGTH OF STRING
        MUL R17, R16

        ADD ZL, R0
        ADC ZH, R1

L1:     LPM R16, Z+
        ST X+, R16
        DEC R17
        TST R17
        BRNE L1
.ENDM
```

Searching For a String

- List contains 20 names, each is 16 characters, null-terminated long. Find the name “Dave   “,0

```
    LDI R20, 0          ;FOUND = 0

l0: LDI ZH, HIGH(2*list)
    LDI ZL, HIGH(2*list)

    LDI R16, 0          ;ix for list
    LDI R21, listlen

    MUL R16,R21         ;ix*16 => R1:R0

    ADD ZL, R0
    ADC ZH, R1          ;Z -> LIST[IX]

    PUSH ZH
    PUSH ZL


    LDI R17, 0          ;ix for strings
l1: LDI ZH, HIGH(2*str1)
    LDI ZL, HIGH(2*str1)
    ADD ZL, R17
    LPM R18, Z          ;str1[ix]

    POP ZL
    POP ZH                          ;Z -> LIST
```

```
    LPM R19, Z+

            PUSH ZH
            PUSH ZL

            CP R18, R19
            BRNE NEXT

            INC R17
            CPI R17, strlen-1
            BRNE L1

            RJMP FOUND

NEXT:       INC R16
            CPI R16, listlen-1
            BRNE l0

            RJMP NOTFOUND


FOUND:      LDI R20, 1
NOTFOUND:
```

List Sorting

- Ascending sort: organize strings alphabetically
  - Example of unorganized list:

    LIST: .DB “Orange ”, “Apple ”, “Peach “, “Pear “, “Banana ”, “Coconut”
  - Sorted:

    LIST: .DB “Apple ”, “Banana ”, “Coconut”, “Orange ”, “Peach “, “Pear “,
- Sort must be performed in DM
  - If list in PM, copy the list into DM then sort

Example: Sequential Sort

- Given a LIST with N strings, each string is M characters
- Algorithm:

```
for i = 0 to N-2
    for j = i+1 to N-1
        str1 = List[i:i+M]
        str2 = List[j:j+M]
        If str1 > str2
            swap strings i,j
```