2.13 Communicating with people

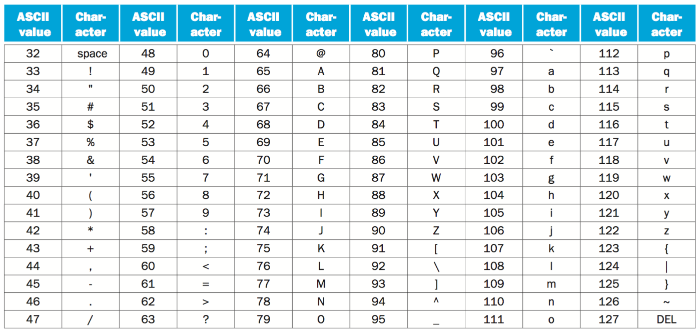
This section has been set as optional by your instructor.

> !(@ |= > (wow open tab at bar is great)
> *Fourth line of the keyboard poem "Hatless Atlas", 1991 (some give names to ASCII characters: "!" is "wow", "(" is open, "|" is bar, and so on).*

Computers were invented to crunch numbers, but as soon as they became commercially viable they were used to process text. Most computers today offer 8-bit bytes to represent characters, with the ***American Standard Code for Information Interchange*** (***ASCII***) being the representation that nearly everyone follows. The following figure summarizes ASCII.

Figure 2.13.1: ASCII representation of characters (COD Figure 2.16).

Note that upper- and lowercase letters differ by exactly 32; this observation can lead to shortcuts in checking or changing upper- and lowercase. Values not shown include formatting characters. For example, 8 represents a backspace, 9 represents a tab character, and 13 a carriage return. Another useful value is 0 for null, the value the programming language C uses to mark the end of a string.



ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter
32	space	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	DEL

Example 2.13.1: ASCII versus binary numbers.

We could represent numbers as strings of ASCII digits instead of as integers. How much does storage increase if the number 1 billion is represented in ASCII versus a 32-bit integer?

Answer

One billion is 1,000,000,000, so it would take 10 ASCII digits, each 8 bits long. Thus the storage expansion would be (10 × 8)/32 or 2.5. Beyond the expansion in storage, the hardware to add, subtract, multiply, and divide such decimal numbers is difficult and would consume more energy. Such difficulties explain why computing professionals are raised to believe that binary is natural and that the occasional decimal computer is bizarre.

PARTICIPATION ACTIVITY 2.13.1: ASCII bit codes (and decimal number equivalents).

Type a character: A   ASCII bit code: **1000001**   ASCII number: **65**

PARTICIPATION ACTIVITY 2.13.2: ASCII.

1) What is the ASCII decimal value for a lower-case 'a'?

Check   **Show answer**

2) What is the ASCII decimal value for an exclamation point?

Check   **Show answer**

3) What two-letter word does this pair of decimal values represent in ASCII? Pay attention to upper/lower case. Use the above ASCII table.
72 105

Check   **Show answer**

4) What two digits does this pair of decimal values represent in ASCII? Use the above ASCII table.
49 50

Check Show answer

A series of instructions can extract a byte from a doubleword, so load register and store register are sufficient for transferring bytes as well as words. Because of the popularity of text in some programs, however, LEGv8 provides instructions to move bytes. *Load byte* (`LDURB`) loads a byte from memory, placing it in the rightmost 8 bits of a register. *Store byte* (`STURB`) takes a byte from the rightmost 8 bits of a register and writes it to memory. Thus, we copy a byte with the sequence

```
LDURB X9,[X0,#0]   // Read byte from source
STURB X9,[X1,#0]   // Write byte to destination
```

PARTICIPATION ACTIVITY 2.13.3: LDURB and STURB.

Assume memory at the indicated addresses have the given decimal values:
2000: 65
2001: 66
2002: 67
2003: 68
2004: 69
2005: 70
2006: 71
2007: 72

1) If the memory values represent a string of characters in ASCII, what are those letters?
- abcdefgh
- ABCDEFGH
- 6566676869707172

2) If X1 holds 2000, what is X2 (in decimal) after:
`LDURB X2, [X1, #0]`
- 2000
- 65
- Low byte is 65; rest unknown.

3) If X1 holds 2000 and is increased by 1, what is X2 (in decimal) after:
`LDURB X2, [X1, #0]`
- 65
- 66
- 67

4) If X1 holds 2000, what is X2 (in decimal) after:
`LDUR X2, [X1, #0]`
- 65
- 65 + 66 + 67 + 68 + 69 + 70 + 71 + 72
- A very large number.

5) Suppose X1 holds 3000, and X2 holds 97 (in decimal). Suppose writing to address 3000 prints a character to the screen. What is printed after the following instruction?
`STURB X2, [X1,#0]`
- 97
- a
- 3000

Characters are normally combined into strings, which have a variable number of characters. There are three choices for representing a string: (1) the first position of the string is reserved to give the length of a string, (2) an accompanying variable has the length of the string (as in a structure), or (3) the last position of a string is indicated by a character used to mark the end of a string. C uses the third choice, terminating a string with a byte whose value is 0 (named null in ASCII). Thus, the string "Cal" is represented in C by the following 4 bytes, shown as decimal numbers: 67, 97, 108, and 0. (As we shall see, Java uses the first option.)

Example 2.13.2: Compiling a string copy procedure, showing how to use C strings.

The procedure `strcpy` copies string `y` to string `x` using the null byte termination convention of C:

```
void strcpy (char x[], char y[])
{
   size t i;
```

```
    i = 0;
    while ((x[i] = y[i]) != '\0') /* copy & test byte */
    i += 1;
}
```

What is the LEGv8 assembly code?

Answer

Below is the basic LEGv8 assembly code segment. Assume that base addresses for arrays `x` and `y` are found in `X0` and `X1`, while `i` is in `X19`. `strcpy` adjusts the stack pointer and then saves the saved register `X19` on the stack:

```
strcpy:
    SUBI   SP, SP, #8    // adjust stack for 1 more item
    STUR   X19, [SP,#0]  // save X19
```

To initialize `i` to 0, the next instruction sets `X19` to 0 by adding 0 to 0 and placing that sum in `X19`:

```
ADD   X19, XZR, XZR      // i = 0 + 0
```

This is the beginning of the loop. The address of `y[i]` is first formed by adding `i` to `y[]`:

```
L1: ADD   X10, X19, X1   // address of y[i] in X10
```

Note that we don't have to multiply `i` by 8 since `y` is an array of *bytes* and not of doublewords, as in prior examples.

To load the character in `y[i]`, we use load byte unsigned, which puts the character into `X11`:

```
LDURB   X11, [X10,#0]   // X11 = y[i]
```

A similar address calculation puts the address of `x[i]` in `X12`, and then the character in `X11` is stored at that address.

```
ADD     X12, X19, X0    // address of x[i] in X12
STURB   X11, [X12,#0]   // x[i] = y[i]
```

Next, we exit the loop if the character was 0. That is, we exit if it is the last character of the string:

```
CBZ   X11, L2           // if y[i] == 0, go to L2
```

If not, we increment `i` and loop back:

```
ADDI   X19, X19, #1     // i = i + 1
B      L1               // go to L1
```

If we don't loop back, it was the last character of the string; we restore `X19` and the stack pointer, and then return.

```
L2: LDUR   X19, [SP,#0] // y[i] == 0: end of string.
                        // Restore old X19
    ADDI   SP, SP, #8   // pop 1 doubleword off stack
    BR     LR           // return
```

String copies usually use pointers instead of arrays in C to avoid the operations on `i` in the code above. See COD Section 2.14 (Arrays versus Pointers) for an explanation of arrays versus pointers.

Since the procedure `strcpy` above is a leaf procedure, the compiler could allocate `i` to a temporary register and avoid saving and restoring `X19`. Hence, instead of thinking of these registers as being just for temporaries, we can think of them as registers that the callee should use whenever convenient. When a compiler finds a leaf procedure, it exhausts all temporary registers before using registers it must save.

PARTICIPATION ACTIVITY 2.13.4: String copy example.

Consider the above example of a C string copy procedure.

1) In C, `(x[i] = y[i])` not only assigns x[i] with y[i], but also evaluates the value that was assigned.
   - Yes
   - No

2) In the C code, the comparison with '\0' (known as null) checks whether this character is the last character in the string.
   - Yes
   - No

3) In the LEGv8 assembly code, the first two instructions (SUBI, STUR) are needed because `strcpy` is a procedure.
   - Yes
   - No

4) If X1 holds the value 5000, what is the value loaded into X1 the first time this statement is executed:
   `L1: ADD X1, X19, X1`
   - 5000

○ 5001

5) If X1 holds 5000, would LDUR work the same as LDURB in the assembly code?

○ Yes

○ No

6) LDURB is used to load a value into array y, and again to load a value into array x.

○ Yes

○ No

7) The `B L1` instruction executes if y's current character is 0.

○ Yes

○ No

8) The instructions at L2 restore X19's value, adjust the stack pointer, and return to the caller program.

○ Yes

○ No

Characters and strings in Java

Unicode is a universal encoding of the alphabets of most human languages. The figure below gives a list of Unicode alphabets; there are almost as many *alphabets* in Unicode as there are useful *symbols* in ASCII. To be more inclusive, Java uses Unicode for characters. By default, it uses 16 bits to represent a character.

Figure 2.13.2: Example alphabets in Unicode (COD Figure 2.17).

Unicode version 4.0 has more than 160 "blocks", which is their name for a collection of symbols. Each block is a multiple of 16. For example, Greek starts at 0370_{hex}, and Cyrillic at 0400_{hex}. The first three columns show 48 blocks that correspond to human languages in roughly Unicode numerical order. The last column has 16 blocks that are multilingual and are not in order. A 16-bit encoding, called UTF-16, is the default. A variable-length encoding, called UTF-8, keeps the ASCII subset as eight bits and uses 16 or 32 bits for the other characters. UTF-32 uses 32 bits per character. To learn more, see www.unicode.org.

Latin	Malayalam	Tagbanwa	General Punctuation
Greek	Sinhala	Khmer	Spacing Modifier Letters
Cyrillic	Thai	Mongolian	Currency Symbols
Armenian	Lao	Limbu	Combining Diacritical Marks
Hebrew	Tibetan	Tai Le	Combining Marks for Symbols
Arabic	Myanmar	Kangxi Radicals	Superscripts and Subscripts
Syriac	Georgian	Hiragana	Number Forms
Thaana	Hangul Jamo	Katakana	Mathematical Operators
Devanagari	Ethiopic	Bopomofo	Mathematical Alphanumeric Symbols
Bengali	Cherokee	Kanbun	Braille Patterns
Gurmukhi	Unified Canadian Aboriginal Syllabic	Shavian	Optical Character Recognition
Gujarati	Ogham	Osmanya	Byzantine Musical Symbols
Oriya	Runic	Cypriot Syllabary	Musical Symbols
Tamil	Tagalog	Tai Xuan Jing Symbols	Arrows
Telugu	Hanunoo	Yijing Hexagram Symbols	Box Drawing
Kannada	Buhid	Aegean Numbers	Geometric Shapes

The LEGv8 instruction set has explicit instructions to load and store such 16- bit quantities, called *halfwords*. *Load half* (`LDURH`) loads a halfword from memory, placing it in the rightmost 16 bits of a register. Like load byte, *load half* (`LDURH`) treats the halfword as a signed number and thus sign-extends to fill the 48 leftmost bits of the register. *Store half* (`STURH`) takes a halfword from the rightmost 16 bits of a register and writes it to memory. We copy a halfword with the sequence

```
LDURH X19, [X0,#0] // Read halfword (16 bits) from source
STURH X9, [X1,#0]  // Write halfword (16 bits) to dest.
```

Strings are a standard Java class with special built-in support and predefined methods for concatenation, comparison, and conversion. Unlike C, Java includes a word that gives the length of the string, similar to Java arrays.

Elaboration

ARMv8 software is required to keep the stack aligned to "quadword" (16 byte) addresses to get better performance. This convention means that a `char` *variable allocated on the stack occupies 16 bytes, even though it needs less. However, a C string variable or an array of bytes will pack 16 bytes per quadword, and a Java string variable or array of shorts packs 8 halfwords per quadword.*

Elaboration

Reflecting the international nature of the web, most web pages today use Unicode instead of ASCII. Hence, Unicode may be even more popular than ASCII today.

Elaboration

*LEGv8 keeps everything 64 bits vs. providing both 32-bit and 64-bit address instructions as in ARMv8, which means it needs to include **`STURW`** (store word) as an instruction even though it is not specified in ARMv8 in assembly language. ARMv8 just uses **`STUR`** with a W register name (32-bit register) instead of X register name (64-bit register).*

PARTICIPATION ACTIVITY 2.13.5: Check yourself: Character and string representation.

1) Strings are just an informal name for single-dimension arrays of characters in C and Java.
   - ○ True
   - ○ False

2) A string in C takes about half the memory as the same string in Java.
   - ○ True
   - ○ False

3) Strings in C and Java use null (0) to mark the end of a string.
   - ○ True
   - ○ False

4) Operations on strings, like length, are faster in C than in Java.
   - ○ True
   - ○ False

PARTICIPATION ACTIVITY 2.13.6: Check yourself: Memory requirements.

1) Which type of variable that can contain $1,000,000,000_{ten}$ takes the most memory space?
   - ○ int in C
   - ○ string in C
   - ○ string in Java