

THE ASCII TABLE

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	:	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

USING THE HEX COLUMN, CONVERT THE FOLLOWING HEX DATA TO CHARACTER

(i.e text) DATA: See Figure 1.2 in the book for difference between \n (one character) and \ then n (two characters)

#	i	n	c	l	u	d	e	sp	<	s	t	d	i	o
23	69	6E	63	6C	75	64	65	20	3C	73	74	64	69	6F

.	h	>	\n	m	a	i	n	(v	o	i	d)
2E	68	3E	A	6D	61	69	6E	28	76	6F	69	64	29

sp	{	\n	sp	sp	sp	p	r	i	n	t	f	("	G
20	7B	A	20	20	20	70	72	69	6E	74	66	28	22	47

o	sp	B	u	c	k	s	!	sp	\n	")	;	}	\n
6F	20	42	75	63	6B	73	21	20	A	22	29	3B	7D	A

What does it say?

```
#include <stdio.h>
main(void) {
    printf("Go Bucks! \n");
}
```

The actual representation/storage of bytes for the \n is two bytes but the interpretation is one byte similar to any new line character.

Basic optimizations

```
% gcc -S hellob.c
% more hellob.s

.file "hellob.c"
.section.rodata

.LC0:
.string "Go Bucks! O-H "
.text

.globl main
.type main,@function

main:
.LFB0:
.cfi_startproc

pushq %rbp

.cfi_def_cfa_offset 16
.cfi_offset 6, -16

movq %rsp,%rbp

.cfi_def_cfa_register 6

movl $.LC0,%edi
call puts
leave

.cfi_def_cfa 7, 8

ret

.cfi_endproc

.LFE0:
.size main,.-main
.ident "GCC: (GNU) 4.4.6 20120305 (Red Hat 4.4.6-4)"
.section.note.GNU-stack,"",@progbits
```

```
% gcc -O1 -S hellob.c
% more hellob.s

.file "hellob.c"
.section.rodata.str1.1,"aMS",@progbits,1

.LC0:
.string "Go Bucks! O-H "
.text

.globl main
.type main,@function

main:
.LFB11:
.cfi_startproc

subq $8,%rsp

.cfi_def_cfa_offset 16

movl $.LC0,%edi
call puts
addq $8,%rsp

.cfi_def_cfa_offset 8

ret

.cfi_endproc

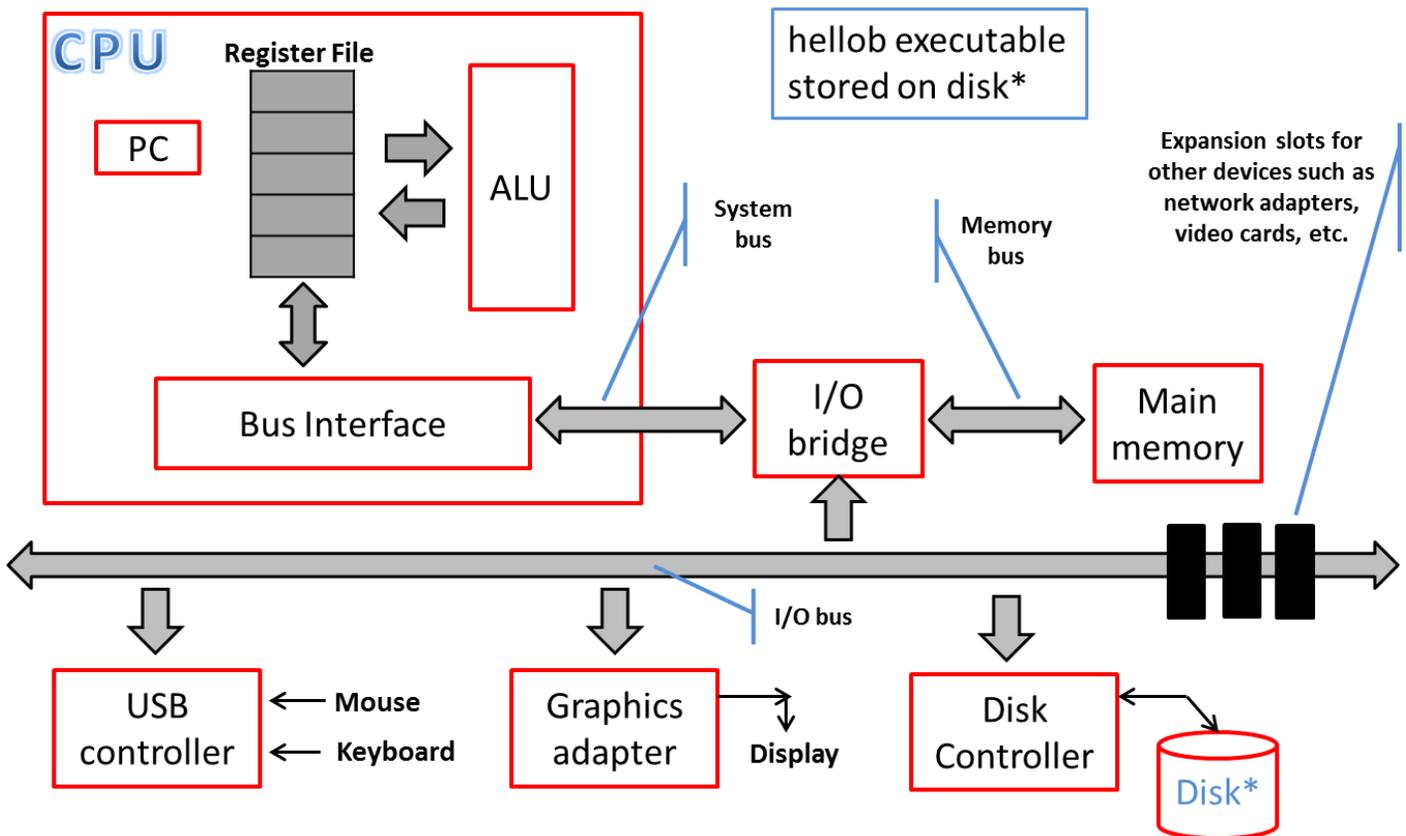
.LFE11:
.size main,.-main
.ident "GCC: (GNU) 4.4.6 20120305 (Red Hat 4.4.6-4)"
.section.note.GNU-stack,"",@progbits
```

32-bit vs 64-bit

```
% gcc -O1 -S -m32 hellob.c
% more hellob.s
.file "hellob.c"
.section
.rodata.str1.1,"aMS",@progbits,1
.LC0:
.string "Go Bucks! O-H "
.text
.globl main
.type main,@function
main:
pushl %ebp
movl %esp,%ebp
andl $-16,%esp
subl $16,%esp
movl $.LC0,(%esp)
call puts
leave
ret
.size main,.-main
.ident "GCC: (GNU) 4.4.6 20120305 (Red Hat 4.4.6-4)"
.section.note.GNU-stack,"",@progbits
```

TRY:

```
gcc -E -P hellob.c → to see some of the .i file info; scrolls on the screen <stdio.h>
look for "program" at the very very end!
gcc -c hellob.c → .o file
gcc -S hellob.c → .s file
```



1. Reading the “hellob” command from the keyboard – the CPU is making the read request.
 - a. User types in “hellob” from the keyboard
 - b. Characters stored in a register.
 - i. Why here first? Why not straight to memory?
 - c. “hellob” is stored in main memory
2. Once enter key is pressed, the executable/object file is loaded from disk into memory
 - a. Disk controller read and transfers
 - b. Does not pass through the CPU (DMA)
 - c. Disk controller sends an interrupt to the CPU to notify
3. Writing the output string from memory to the display
 - a. Machine language instructions are executed starting with the main routine
 - b. Instructions copy string bytes from memory to register file
 - c. String bytes then output to display

CHECKING THE SIZE OF C DATA TYPES – sizeck.c

```
#include <stdio.h>
void main()
{
    // integers
    printf("size of char is %d\n",sizeof(char));
    printf("size of short int is %d\n",sizeof(short));    // short int
    printf("size of int is %d\n",sizeof(int));
    printf("size of long is %d\n",sizeof(long int));    // long
    printf("size of long long is %d\n\n",sizeof(long long));

    // real
    printf("size of float is %d\n",sizeof(float));
    printf("size of double is %d\n",sizeof(double));
    printf("size of long double is %d\n\n",sizeof(long double));

    // unsigned integers
    printf("size of unsigned char is %d\n",sizeof(unsigned char));
    printf("size of unsigned short int is %d\n",sizeof(unsigned short int));
    printf("size of unsigned int is %d\n",sizeof(unsigned int));
    printf("size of unsigned long int is %d\n",sizeof(unsigned long int));

    // pointers
    printf("size of char * is %d\n",sizeof(char *));
    printf("size of int * is %d\n",sizeof(int *));
    printf("size of double * is %d\n",sizeof(double *));
}
```

“sizeof” function returns
number of bytes

OUTPUT regular gcc command:

size of char is 1
size of short int is 2
size of int is 4
size of long is 8
size of long long is 8

size of float is 4
size of double is 8
size of long double is 16

size of unsigned char is 1
size of unsigned short int is 2
size of unsigned int is 4
size of unsigned long int is 8

size of char * is 8
size of int * is 8
size of double * is 8

OUTPUT –m32 option:

size of char is 1
size of short int is 2
size of int is 4
size of long is 4
size of long long is 8

size of float is 4
size of double is 8
size of long double is 12

size of unsigned char is 1
size of unsigned short int is 2
size of unsigned int is 4
size of unsigned long int is 4

size of char * is 4
size of int * is 4
size of double * is 4

INTEGER and FLOATING POINT ARITHMETIC

EXAMPLE 1 – overflow.c

```
#include <stdio.h>
#include <float.h>
void main()
{
    // integer arithmetic overflow
    int a = 200 * 300 * 400 * 500;    // 12,000,000,000
    printf("a = %d\n",a);

    // floating-point overflow?
    double d = DBL_MAX;
    double e =DBL_MAX;
    printf("\ndouble d max is %g", d);
    d = d + 1; // change 1 to 100
    printf("\nadd one to DBL_MAX to get %g",d);
    printf("\nadd max to max to get %g",d+e);
}
```

```
OUTPUT:
% gcc -o overflow overflow.c
overflow.c: In function 'main':
overflow.c:8: warning: integer overflow in
expression

% overflow
a = -884901888
double d max is 1.79769e+308
add one to DBL_MAX to get 1.79769e+308
add max to max to get inf
```

EXAMPLE 2 – floatpt.c

```
#include <stdio.h>
#include <float.h>
void main()
{
    float a, b, c; // float to double
    a = 0.056;
    b = 0.064;
    c = b - a; // difference is 0.008
```

```
OUTPUT:
a=0.056000, b=0.064000 and c=0.008000
0.064000 - 0.056000 != 0.008000 Subtraction has round-off error
x = 0.000000
y with fspec = 3.140000
y with espec = 3.140000e+00
```

```
printf("a=%f, b=%f and c=%f\n",a,b,c);
// printf("a=%f, b=%f and c=%.16f\n",a,b,c); // using width specifier
```

```
if( b - a == 0.008)
    printf("%f - %f == %f subtraction is correct",b,a,b-a);
else
    printf("%f - %f != %f Subtraction has round-off error",b,a,b-a);
```

```
float x = (3.14 + 1e20) - 1e20;    // parens not necessary
printf("x = %f\n",x);
```

```
float y = 3.14 + (1e20 - 1e20);
printf("y with fspec = %f\n",y);
printf("y with espec = %e\n",y);    // notice %e for sci not }
```

The Secret Message

Use the provided information to complete the binary matrix. The binary digits when read across (left to right), or down (top to bottom), are to represent the same values as the decimal, hexadecimal and octal representations provided. You can cross check your answers with either the horizontal or vertical answers.

When you have completed the binary matrix, blacken all the squares containing 1s to obtain a secret message!

MSB																		LSB															DEC	HEX	OCT
0	1	1	1	0	1	1	1	0	1	1	1	0	1	0	0				30580	7774	73564														
0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0				17748		42524														
0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0				17748	4554															
0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1				30583		73567														
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						0														
1	0	1	0	1	0	1	0	1	0	1	0	1	1	1	0				43694																
1	1	1	0	1	0	1	0	1	1	1	0	0	0	0	1					EAE2	165342														
1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0					AAA0															
1	0	1	0	1	1	1	0	1	0	1	0	0	1	0	0				44708		127244														

15		303	288		481					303	480	8		44	32	<u>DEC</u>
			120	F	1E1	12F	1E0	F			1E0		1E9		20	<u>HEX</u>
17	744	457				457	740	17		457		10	751		40	<u>OCT</u>

Here are standard base conversion tables to help you

DEC	OCT	HEX	BIN	Notes
	0	0	0	-
1	1	1	1	2 ⁰
2	2	2	10	2 ¹
3	3	3	11	
4	4	4	100	2 ²
5	5	5	101	
6	6	6	110	
7	7	7	111	
8	10	8	1000	2 ³
9	11	9	1001	
10	12	A	1010	
11	13	B	1011	
12	14	C	1100	
13	15	D	1101	
14	16	E	1110	
15	17	F	1111	
16	20	10	10000	2 ⁴
17	21	11	10001	
18	22	12	10010	
19	23	13	10011	
20	24	14	10100	

DEC	OCT	HEX	BIN	Notes
	0	0	0	
32	40	20	100000	2 ⁵
64	100	40	1000000	2 ⁶
128	200	80	10000000	2 ⁷
256	400	100	100000000	2 ⁸
512	1000	200	1000000000	2 ⁹
1,024	2000	400	10000000000	2 ¹⁰ (≈10 ³)
2,048	4000	800	100000000000	2 ¹¹
4,096	10000	1000	1000000000000	2 ¹²
8,192	20000	2000	10000000000000	2 ¹³
16,384	40000	4000	100000000000000	2 ¹⁴
32,768	100000	8000	1000000000000000	2 ¹⁵
65,536	200000	10000	10000000000000000	2 ¹⁶
1,048,576	4000000	100000	**	2 ²⁰ (≈10 ⁶)
1,073,741,824	10000000000	40000000	**	2 ³⁰ (≈10 ⁹)
2,147,483,648	200000000000	800000000	**	2 ³¹
4,294,967,296	4000000000000	10000000000	**	2 ³²

From the unix command prompt: %objdump -D -t -s hellob

```
#include <stdio.h> // hellob.c
main(void)
{
    int x = 1234567; // is 0012d687 in hex
    char cry[] = "Go Bucks!";
    printf("Go Bucks! O-H \n");
}
```

SYMBOL TABLE:

```
.
.
.
08048400 g F .text 0000005a __libc_csu_init
08049660 g *ABS* 00000000 __bss_start
08049668 g *ABS* 00000000 _end
00000000 F *UND* 00000000 puts@@GLIBC_2.0
08049660 g *ABS* 00000000 _edata
0804845a g F .text 00000000 .hidden __i686.get_pc_thunk.bx
080483b4 g F .text 00000036 main
08048290 g F .init 00000000 _init.
.
.
.
```

80483b4 <main>:

```
80483b4: 55 push %ebp
80483b5: 89 e5 mov %esp,%ebp
80483b7: 83 e4 f0 and $0xffffffff0,%esp
80483ba: 83 ec 20 sub $0x20,%esp
80483bd: c7 44 24 1c 87 d6 12 00 movl $0x12d687,0x1c(%esp)
80483c5: c7 44 24 12 47 6f 20 42 movl $0x42206f47,0x12(%esp)
80483cd: c7 44 24 16 75 63 6b 73 movl $0x736b6375,0x16(%esp)
80483d5: 66 c7 44 24 1a 21 00 movw $0x21,0x1a(%esp)
80483dc: c7 04 24 b4 84 04 08 movl $0x80484b4,(%esp)
80483e3: e8 08 ff ff ff call 80482f0 <puts@plt>
80483e8: c9 leave
80483e9: c3 ret
```

The value of x is stored in hex, little endian style.; and since it's an integer, it's 4 bytes long.

"Go Bucks!" has a length of 9 plus the '\0' end of string terminator symbol for a total length of **10 characters.**

INTEGER REPRESENTATIONS

Unsigned Representation (B2U). Two's Complement (B2T). Signed Magnitude (B2S). Ones' Complement (B2O).

scheme	size in bits (w)	minimum value	maximum value
B2U	5	0	31
B2U	8	0	255
B2U	12	0	4095
B2U	16	0	65535
B2T	5	-16	15
B2T	8	-128	127
B2T	12	-2048	2047
B2T	16	-32768	32767
B2O	5	-15	15
B2O	8	-127	127
B2O	12	-2047	2047
B2O	16	-32767	32767
B2S	5	-15	15
B2S	8	-127	127
B2S	12	-2047	2047
B2S	16	-32767	32767

Remember:

2^{15} 2^{14} 2^{13} 2^{12} 2^{11} 2^{10} 2^9 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
 32768 16384 8192 4096 2048 1024 512 256 128 64 32 16 8 4 2 1

1. Convert 10111 base 2 to decimal for the following decoding schemes:

B2U = _____ 23 B2O = _____ -8

B2T = _____ -9 B2S = _____ -7

2. Convert the number 95 base 10 to the following 8-bit encoding schemes: 0101 1111 all the same

_____ B2U _____ B2T

_____ B2S _____ B2O

3. Convert the number 130 base 10 to the following 8-bit encoding schemes: 1000 0010 for B2U; rest=overflow

_____ B2U _____ B2T

_____ B2S _____ B2O

4. Convert the number -223 base 10 to the following 16-bit encoding schemes:

_____ B2U no negative #s

_____ B2T 0xFF21

_____ B2O 0xFF20

_____ B2S 0x80DF

SIGNED/UNSIGNED

```
#include <stdio.h>
void main()
{
    int x;
    for (x = -8; x < 8; x++)
        printf("xd = %d xu = %u xx = %.4x\n", x, (unsigned short) x, (unsigned short) x);

    printf("\n");

    for (x = -8; x < 8; x++)
        printf("xd = %d xu = %u xx = %.8x\n", x, x, x);

    x = 65528;
    printf("\n65528 in hex = %.8x\n", x);

    short a = 0xFFFF + 1;
    printf("\n0xFFFF + 1 = %d\n", a);
}
```

```
% gcc -o sign2unsign sign2unsign.c
sign2unsign.c: In function 'main':
sign2unsign.c:17: warning: overflow in implicit constant conversion
```

```
xd = -8 xu = 65528 xx = fff8
xd = -7 xu = 65529 xx = fff9
xd = -6 xu = 65530 xx = fffa
xd = -5 xu = 65531 xx = fffb
xd = -4 xu = 65532 xx = fffc
xd = -3 xu = 65533 xx = fffd
xd = -2 xu = 65534 xx = fffe
xd = -1 xu = 65535 xx = ffff
xd = 0 xu = 0 xx = 0000
xd = 1 xu = 1 xx = 0001
xd = 2 xu = 2 xx = 0002
xd = 3 xu = 3 xx = 0003
xd = 4 xu = 4 xx = 0004
xd = 5 xu = 5 xx = 0005
xd = 6 xu = 6 xx = 0006
xd = 7 xu = 7 xx = 0007

xd = -8 xu = 4294967288 xx = ffffffff8
xd = -7 xu = 4294967289 xx = ffffffff9
xd = -6 xu = 4294967290 xx = ffffffff10
xd = -5 xu = 4294967291 xx = ffffffff11
xd = -4 xu = 4294967292 xx = ffffffff12
xd = -3 xu = 4294967293 xx = ffffffff13
xd = -2 xu = 4294967294 xx = ffffffff14
xd = -1 xu = 4294967295 xx = ffffffff15
xd = 0 xu = 0 xx = 00000000
xd = 1 xu = 1 xx = 00000001
xd = 2 xu = 2 xx = 00000002
xd = 3 xu = 3 xx = 00000003
xd = 4 xu = 4 xx = 00000004
xd = 5 xu = 5 xx = 00000005
xd = 6 xu = 6 xx = 00000006
xd = 7 xu = 7 xx = 00000007

65528 in hex = 0000fff8

0xFFFF + 1 = 0
```