Appendices

Base Conversion Table and Powers of two

Decimal	Hexadecimal	Binary	2^n Hex	2^n Decimal
0	0	0000	1	1
1	1	0001	2	2
2	2	0010	4	4
3	3	0011	8	8
4	4	0100	10	16
5	5	0101	20	32
6	6	0110	40	64
7	7	0111	80	128
8	8	1000	100	256
9	9	1001	200	512
10	a	1010	400	1024
11	b	1011	800	2048
12	c	1100	1000	4096
13	d	1101	2000	8192
14	e	1110	4000	16384
15	f	1111	8000	32768

Hexadecimal Addition Table

+	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10
2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11
3	4	5	6	7	8	9	a	b	c	d	e	f	10	11	12
4	5	6	7	8	9	a	b	c	d	e	f	10	11	12	13
5	6	7	8	9	a	b	c	d	e	f	10	11	12	13	14
6	7	8	9	a	b	c	d	e	f	10	11	12	13	14	15
7	8	9	a	b	c	d	e	f	10	11	12	13	14	15	16
8	9	a	b	c	d	e	f	10	11	12	13	14	15	16	17
9	a	b	c	d	e	f	10	11	12	13	14	15	16	17	18
a	b	c	d	e	f	10	11	12	13	14	15	16	17	18	19
b	c	d	e	f	10	11	12	13	14	15	16	17	18	19	1a
c	d	e	f	10	11	12	13	14	15	16	17	18	19	1a	1b
d	e	f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c
e	f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d
f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e

Hexadecimal Multiplication Table

*	2	3	4	5	6	7	8	9	a	b	c	d	e	f
2	4	6	8	a	c	e	10	12	14	16	18	1a	1c	1e
3	6	9	c	f	12	15	18	1b	1e	21	24	27	2a	2d
4	8	c	10	14	18	1c	20	24	28	2c	30	34	38	3c
5	a	f	14	19	1e	23	28	2d	32	37	3c	41	46	4b
6	c	12	18	1e	24	2a	30	36	3c	42	48	4e	54	5a
7	e	15	1c	23	2a	31	38	3f	46	4d	54	5b	62	69
8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	12	1b	24	2d	36	3f	48	51	5a	63	6c	75	7e	87
a	14	1e	28	32	3c	46	50	5a	64	6e	78	82	8c	96
b	16	21	2c	37	42	4d	58	63	6e	79	84	8f	9a	a5
c	18	24	30	3c	48	54	60	6c	78	84	90	9c	a8	b4
d	1a	27	34	41	4e	5b	68	75	82	8f	9c	a9	b6	c3
e	1c	2a	38	46	54	62	70	7e	8c	9a	a8	b6	c4	d2
f	1e	2d	3c	4b	5a	69	78	87	96	a5	b4	c3	d2	e1

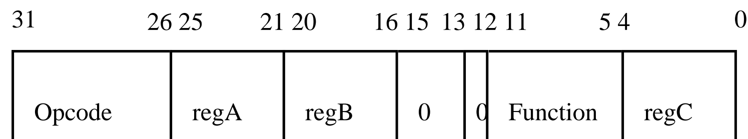
The ASCII character set

00	NUL	01	SOH	02	STX	03	ETX
04	EOT	05	ENQ	06	ACK	07	BEL (\a)
08	BS (\b)	09	HT (\t)	0A	LF (\n)	0B	VT (\v)
0C	FF (\f)	0D	CR (\r)	0E	SO	0F	SI
10	DLE	11	DC1	12	DC2	13	DC3
14	DC4	15	NAK	16	SYN	17	ETB
18	CAN	19	EM	1A	SUB	1B	ESC
1C	FS	1D	GS	1E	RS	1F	US
20	SP	21	!	22	"	23	#
24	$	25	%	26	&	27	'
28	(	29	)	2A	*	2B	+
2C	,	2D	-	2E	.	2F	/
30	0	31	1	32	2	33	3
34	4	35	5	36	6	37	7
38	8	39	9	3A	:	3B	;
3C	<	3D	=	3E	>	3F	?
40	@	41	A	42	B	43	C
44	D	45	E	46	F	47	G
48	H	49	I	4A	J	4B	K
4C	L	4D	M	4E	N	4F	O
50	P	51	Q	52	R	53	S
54	T	55	U	56	V	57	W
58	X	59	Y	5A	Z	5B	[
5C	\	5D	]	5E	^	5F	_
60	`	61	a	62	b	63	c
64	d	65	e	66	f	67	g
68	h	69	i	6A	j	6B	k
6C	l	6D	m	6E	n	6F	o
70	p	71	q	72	r	73	s
74	t	75	u	76	v	77	w
78	x	79	y	7A	z	7B	{
7C	|	7D	}	7E	~	7F	DEL

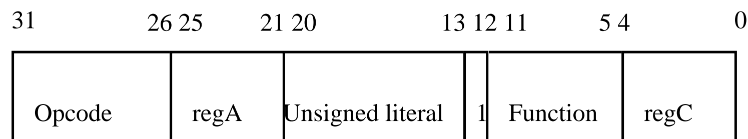
Alpha opcodes and function codes for some common instructions

Name	Format	Opcode	Function code
addq	Operate	0x10	0x20
subq	Operate	0x10	0x29
ldq	Memory	0x29	
stq	Memory	0x2d	
beq	Branch	0x39	
bne	Branch	0x3d	

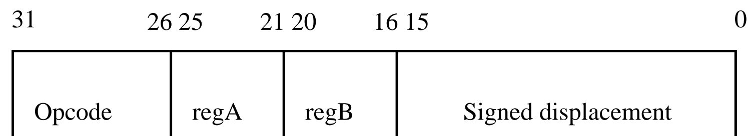
Alpha Instruction Formats



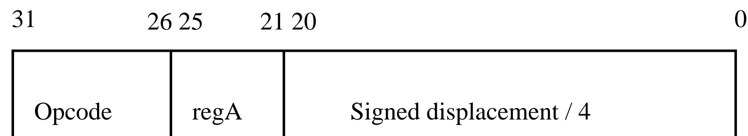
Integer operate instruction with second operand a register



Integer operate instruction with second operand a literal



Memory access instruction



Branch instruction

Alpha Registers

0	v0
1-8	t0-t7
9-14	s0-s5
15	fp
16-21	a0-a5
22-25	t8-t11

26	ra
27	pv
28	at
29	gp
30	sp
31	zero

(Note: The register numbers are in decimal).

Commonly used instructions

Integer operate instructions

```
Opcode $regA, $regB, $regC      //   regC = regA op regB
Opcode $regA, constantB, $regC  //   regC = regA op constantB
```

Arithmetic integer operate instructions

addq	add	+
subq	subtract	-
mulq	multiply	*
divq/divqu	divide, signed/unsigned	/
modq/modqu	modulo, signed/unsigned	%
s8addq	scaled 8 add	8*operandA+operandB

Shift integer operate instructions

sll	shift left logical	<<
srl	shift right logical	>>>
sra	shift right arithmetic	>>

Compare integer operate instructions

cmpeq	compare equal	==
cmplt/cmpult	compare less than signed/unsigned	<
cmple/cmpule	compare less than or equal signed/unsigned	<=

Logical integer operate instructions

<table>
<tr><td>and</td><td>and</td><td>&</td></tr>
<tr><td>bic</td><td>bit clear</td><td>& ~</td></tr>
<tr><td>bis/or</td><td>bit set/or</td><td>|</td></tr>
<tr><td>eqv/xornot</td><td>equivalent/exclusive or not</td><td>^ ~</td></tr>
<tr><td>ornot</td><td>or not</td><td>| ~</td></tr>
<tr><td>xor</td><td>exclusive or</td><td>^</td></tr>
</table>

Memory instructions

```
Opcode $regA, displacement($regB)
Opcode $regA, ($regB)
Opcode $regA, constant
```

Load address instruction

```
//  regA =  displacement + regB
```

lda	load address

Load memory instructions

```
//  regA =  Memory[ displacement + regB ]
```

ldq	load quadword
ldbu	load byte unsigned

Store memory instructions

```
//  Memory[ displacement + regB ] = regA
```

stq	store quadword
stb	store byte

Branch instructions

Conditional branch instructions

```
Opcode $regA, destination
//  if ( condition holds for regA ) pc = destination
```

beq	branch equal
bne	branch not equal
blt	branch less than
ble	branch less than or equal
bgt	branch greater than
bge	branch greater than or equal
blbs	branch low bit set
blbc	branch low bit clear

Unconditional branch instructions

```
Opcode destination;
```

br	branch
bsr	branch to subroutime

Jump instruction

```
Opcode ($reg);
```

jmp	jump
jsr	jump to subroutine

Return instruction

ret	return

Callpal instruction

```
call_pal constant;
```

call_pal	call PALcode

Pseudoinstructions

Load immediate

```
ldiq $regA, constant            //      regA = constant
```

ldiq	load immediate quadword

Clear

```
clr $regA                       //      regA = 0
```

clr	clear

Unary pseudoinstructions

```
Opcode $regB, $regC             //      regC = op regB
Opcode $constantB, $regC        //      regC = op constantB
```

mov	move
negq	negate

Library functions in the simulator

In block Sys:

- long getChar(). Reads a character from the simple terminal.
- long putChar(char c). Writes a character to the simple terminal.
- void exit(). Causes the process to exit.

In block IO:

- void newline(). Prints a newline.
- void print(char *s). Prints a string.
- void error(char *s). Prints a string then exits.
- void readLine(char *s, long max). Reads a line of input into a buffer, and terminates the text with a null byte. Discards text that will not fit into the buffer.
- void printf(char *s, long param0, long param1, long param2, long param3, long param4). Prints the parameters according to the format string s.

In block Number:

- long fromString(char *buffer, long base). Converts the text in the buffer from the specified base into internal form. If base is 0, determines the base from the start of the text.
- char *toUnsigned(unsigned long value, long base). Converts the unsigned number into a base.
- char *toSigned(long value, long base). Converts the signed number into a base.

In block String:

- char *fromChar(char c). Creates a string containing the character c.
- long compare(char *s, char *t). Compares two strings and indicates their order by a value <, ==, > 0 depending on whether s < t, s == t, s > t., in the usual lexigraphical ordering.
- long length(char *s). Returns the length of the string s.
- void copy(char *s, char *t). Copies the string pointed to by t into the buffer pointed to by s.
- char *padLeft(char *s, char padChar, long fieldWidth). Pads s on the left with the pad character, to create a string of length fieldWidth.
- char *padRight(char *s, char padChar, long fieldWidth). Pads s on the right with the pad character, to create a string of length fieldWidth.

Function that return a string (char *) use static space. The space will be overwritten by a later invocation.