

# Reference Pages

## Instructions and Op codes

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0001				DR			SR1			0	xx		SR2		
ADD	0001				DR			SR1			1	imm5				
AND	0101				DR			SR1			0	xx		SR2		
AND	0101				DR			SR1			1	imm5				
BRx	0000				n	z	p	pgoffset9								
DEBUG	1000				xxxxxxxxxxxx											
JSR	0100				L	xx		pgoffset9								
JSRR	1100				L	xx	BaseR			index6						
LD	0010				DR			pgoffset9								
LDI	1010				DR			pgoffset9								
LDR	0110				DR			BaseR			index6					
LEA	1110				DR			pgoffset9								
NOT	1001				DR			SR			xxxxxx					
RET	1101				xxxxxxxxxxxx											
ST	0011				SR			pgoffset9								
STI	1011				SR			pgoffset9								
STR	0111				SR			BaseR			index6					
TRAP	1111				xxxx			trapvect8								

Figure 1: Formats of instructions

## Syntax of Assembly Language

Instruction		Example	Description
ADD	DR,SR1,SR2	ADD R0,R3,R0	Add
ADD	DR,SR1,imm5	ADD R3,R3,#-1	Add Immediate
AND	DR,SR1,SR2	AND R5,R5,R3	And
AND	DR,SR1,imm5	AND R3,R3,xF	And Immediate
BRx	addr	BRZP x3020	Branch
DEBUG		DEBUG	Debug
JSR	addr	JSR Mult	Jump Subroutine
JMP	addr	JMP ShutDn	Jump
JSRR	BR,index6	JSRR R2,x0	Jump Subroutine Reg-relative
JMPR	BR,index6	JMPR R4,x10	Jump Register-relative
LD	DR,addr	LD Acc,Value	Load
LDI	DR,addr	LDI R0,x3100	Load Indirect
LDR	DR,BR,index6	LDR R0,R4,xA	Load Register-relative
LEA	DR,addr	LEA R0,Msg1	Load Effective Address
NOT	DR,SR	NOT R2,R2	Not
RET		RET	Return
ST	SR,addr	ST R5,ANSWR	Store
STI	SR,addr	STI R3,x3000	Store Indirect
STR	SR,BR,index6	STR R2,R0,Offset	Store Register-relative
TRAP	trapvect8	TRAP x25	Trap

## Trap Vector Table

<u>Number</u>	<u>Name</u>	<u>Description</u>
x21	OUT	Write the character in R0[7:0] to the console.
x22	PUTS	Write the null-terminated string pointed to by R0 to the console.
x23	IN	Print a prompt on the screen and read a single character from the keyboard. The character is copied to the screen and its ASCII code is copied to R0. The high 8 bits of R0 are cleared.
x25	HALT	Halt execution and print a message to the console.
x31	OUTN	Write the value of R0 to the console as a decimal integer.
x33	INN	Print a prompt on the screen and read a decimal number from the keyboard. The number is echoed to the screen and stored in R0. You may place specific requirements on the size, formatting, etc. of the input.
x43	RND	Store a random number in R0.

## Pseudo Ops

Our abstract machine has the following pseudo ops:

Mnemonic	Meaning
.ORIG	Origin
.END	End
.EQU	Equate
.FILL	Numeric Data
.STRZ	Character Data
.BLKW	Reserve Storage
.ENT	Entry Point
.EXT	External Symbol

## Syntax of Records in Object File

1. A Header Record

record position 1: H

record positions 2-7: a 6 character segment name

record positions 8-11: a 4 Hex character value denoting the initial program load address

record positions 12-15: a 4 Hex character value denoting the length of the segment

2. Text Records

record position 1: T

record positions 2-5: a 4 Hex character address at which the information is to be stored

record positions 6-9: Initial value of that address, as a 4 Hex character string

3. End Record

record position 1: E

record positions 2-5: a 4 Hex character address at which execution is to begin

In addition, your group has designed modification records to encode relocation and linking information.

## ASCII Table

sp	x20	0	x30	}	x40	P	x50	'	x60	p	x70
!	x21	1	x31	A	x41	Q	x51	a	x61	q	x71
"	x22	2	x32	B	x42	R	x52	b	x62	r	x72
#	x23	3	x33	C	x43	S	x53	c	x63	s	x73
\$	x24	4	x34	D	x44	T	x54	d	x64	t	x74
%	x25	5	x35	E	x45	U	x55	e	x65	u	x75
&	x26	6	x36	F	x46	V	x56	f	x66	v	x76
'	x27	7	x37	G	x47	W	x57	g	x67	w	x77
(	x28	8	x38	H	x48	X	x58	h	x68	x	x78
)	x29	9	x39	I	x49	Y	x59	i	x69	y	x79
*	x2A	:	x3A	J	x4A	Z	x5A	j	x6A	z	x7A
+	x2B	;	x3B	K	x4B	[	x5B	k	x6B	{	x7B
,	x2C	<	x3C	L	x4C	\	x5C	l	x6C		x7C
-	x2D	=	x3D	M	x4D	]	x5D	m	x6D	}	x7D
.	x2E	>	x3E	N	x4E	^	x5E	n	x6E	~	x7E
/	x2F	?	x3F	O	x4F	_	x5F	o	x6F	del	x7F

## UTF-8 Encoding

Unicode	Octet 1	Octet 2	Octet 3	Octet 4
U+0000–U+007F	0xxxxxxx			
U+0080–U+07FF	110yyyxx	10xxxxxx		
U+0800–U+FFFF	1110yyyy	10yyyyxx	10xxxxxx	
U+10000–U+10FFFF	11110zzz	10zzyyyy	10yyyyxx	10xxxxxx

## Units of Memory Size

Unit	Abbreviation	Equivalent	Equivalent
1 kilobyte	KB	2 <sup>10</sup> bytes	1,024 bytes
1 megabyte	MB	2 <sup>20</sup> bytes	1,048,576 bytes
1 gigabyte	GB	2 <sup>30</sup> bytes	1,073,741,824 bytes
1 terabyte	TB	2 <sup>40</sup> bytes	1,099,511,627,776 bytes
1 petabyte	PB	2 <sup>50</sup> bytes	1,125,899,906,842,624 bytes

# Precedence Matrix for Simple Programming Language

	<i>VAR</i>	<i>BEGIN</i>	<i>END</i>	<i>END.</i>	<i>INTEGER</i>	<i>FOR</i>	<i>READ</i>	<i>WRITE</i>	<i>TO</i>	<i>DO</i>	<i>:</i>	<i>:</i>	<i>,</i>	<i>:=</i>	<i>+</i>	<i>-</i>	<i>*</i>	<i>DIV</i>	<i>(</i>	<i>)</i>	<i>id</i>	<i>int</i>
<b>PROGRAM</b>	=																					
<b>VAR</b>	=									<	<	<									<	<
<b>BEGIN</b>	=	=			<	<	<			<	<	<									<	<
<b>END</b>	>	>								>	>	>										
<b>INTEGER</b>	>									=	>										=	<
<b>FOR</b>										=											=	<
<b>READ</b>																					=	<
<b>WRITE</b>																					=	<
<b>TO</b>	<	>	>		<	<	<		>	>			<	<	<	<	<	<	<	<	<	<
<b>DO</b>	<	>	>		<	<	<		>	>			<	<	<	<	<	<	<	<	<	<
<b>:</b>	>	>	>		<	<	<		>	>	<										<	<
<b>:</b>	>	>	>		<	<	<		>	>	<										<	<
<b>,</b>																					=	<
<b>:=</b>	>	>						=	>	>				<	<	<	<	<	<	<	<	<
<b>+</b>	>	>						>	>	>				<	<	<	<	<	<	<	<	<
<b>-</b>	>	>						>	>	>				<	<	<	<	<	<	<	<	<
<b>*</b>	>	>						>	>	>				<	<	<	<	<	<	<	<	<
<b>DIV</b>	>	>						>	>	>				<	<	<	<	<	<	<	<	<
<b>(</b>		>	>					>	>	>		<		<	<	<	<	<	<	=	<	<
<b>)</b>		>	>					>	>	>		<		<	<	<	<	<	<	=	<	<
<b>id</b>	>	>	>					>	>	>	>	>	=	>	>	>	>	>	>	>	>	
<b>int</b>	>	>	>					>	>	>	>	>	=	>	>	>	>	>	>	>	>	

# Miscellaneous

## Labels

Labels may be up to 6 alphanumeric characters (e.g., they may not include blanks). The first character of a label must be alphabetic, but must not be an “R” or an “x”.

## Operands and Comments

In the “operands and comments” field, you may prohibit the “operand” part from including blanks. Registers are indicated by their explicit name (such as R3). Constants are written either in hexadecimal notation (preceded by a lowercase “x”) or as decimal integers, which may be positive or negative (preceded by “#”).

An imm5 operand must be in the range #-16..#15, or x0..x1F. An addr operand must be in the range #0..#65535, or x0..xFFFF. Note that only the least significant 9 bits of this value are used in the machine code encoding. An index6 operand must be in the range #0..#63, or x0..x3F. A trapvect8 operand must be in the range x0..xFF.

Symbols can be used in place of any operand. The *only* place where a relocatable symbol can be used is in the final operand of the following instructions: BR, JSR, JMP, LD, LDI, LEA, ST, and STI (i.e., replacing an addr operand).

Symbols can be used in place of an explicit register name (such as R3). In this case, the (absolute) symbol must have a value in the range 0 to 7. Symbols can also be used in place of immediate arguments (imm5, index6, and trapvect8). Again, in this case, the (absolute) symbol must have a value in the appropriate range. In the case of imm5, where there are two different ranges, depending on whether a decimal or hex number is used, the symbol value must be in the union of the two ranges (i.e., #-16..x1F).

When a symbol is used as the last argument for ADD or AND instructions, it is always interpreted as an imm5 operand, rather than a source register. That is, the first form of these instructions is used, where bit 5 is set.

For the LD instruction, the addr operand can contain a literal. A literal is denoted by an “=”, followed by a constant (e.g., =x2A, =#-1). A literal causes the assembler to: generate a location for the literal (after the last programmer-defined location), place the value indicated by the operand in that location, and use the address of that location in the instruction. Literals are allowed only with the LD instruction. The value of a literal must be in the range #-32768..#32767 for decimal constants, and the range x0..xFFFF for hex.

For every addr operand, its page number is checked against the page number of the PC when that instruction is executed (i.e., the location counter plus 1 of the instruction in question). If they are not the same, the assembler should flag this as a fatal error.

## Pseudo Ops

- .ORIG** This must be the first non-comment record in the source program. The operand, if present, must be a hex number in the range x0..xFFFF. The operand indicates the absolute address at which the program is to be loaded. If the operand is absent, the program is relocatable. If the program is relocatable, it must fit within one page of memory. The .ORIG statement must have a label, which is the name of the segment.
- .END** Indicates the end of the input program. An optional operand (a hex integer in the range x0..xFFFF or a symbol) indicates the address at which execution is to begin. If no operand is present, execution begins at the first address in the segment.
- .EQU** Equates the symbol in the label field with the “value” of the operand field, essentially creating a constant within the assembler. The operand field can be a previously defined symbol or a constant. The constant can be written either as a decimal integer (preceded with #) or a hex number (preceded with x).
- .FILL** Defines a one-word quantity whose contents is the value of the operand. The operand is either a symbol a (hex or decimal) constant. Decimal constants must be in the range #-32768..#32767, while hex constants must be in the range x0..xFFFF. If a symbol is used, its value must be in the union of these two ranges (ie #-32768..xFFFF). This value is placed by the assembler in the word of memory that the .FILL pseudo-op occupies. The assembler location counter is moved forward one word.
- .STRZ** Defines a block of words to hold the characters of the null-terminated string in the pseudo-ops operand field. The operand string is enclosed in quotation marks. The ASCII code for each character is stored in bits [7:0], while bits [15:8] are cleared. Since a null (x0000) is added at the end, a STRZ pseudo-op whose operand is “test” will occupy 5 words in memory.
- .BLKW** Sets up a block of storage. The number of words in the block must be at least one and at most xFFFF, as indicated by the constant or previously defined absolute symbol in the operand field. This command moves the assembler location counter forward the corresponding number of words. The block of storage is not initialized by this pseudo op.
- .ENT** The operand field of this pseudo-op is a list of symbols that must be defined in the current segment, and are permitted to be referenced by other segments.
- .EXT** The operand field is a list of symbols that may be referenced legitimately by this segment, but are not defined in this segment. The symbols must be defined in some other segment.

The .ORIG and .EQU pseudo op instructions **require** labels. If the operand of .EQU or .BLKW is a symbol, that symbol must be defined earlier in the program. Labels are optional on the .FILL, .STRZ, and .BLKW pseudo ops. No label is allowed on .END pseudo ops. .FILL, .STRZ, and .BLKW require memory allocation while .ORIG, .EQU, and .END do not.