1 Instruction Set

1.1 Register Instructions

Instruction	Signification	Instruction	Signification
integer addition		**integer multiplication**	
ADD a b c	R.a = R.b + R.c	MUL a b c	R.a = R.b * R.c
ADDI a b ic	R.a = R.b + ic	MULI a b ic	R.a = R.b * ic
ADDIU a b uc	R.a = R.b + uc	MULIU a b uc	R.a = R.b * uc
integer subtraction		**integer division**	
SUB a b c	R.a = R.b - R.c	DIV a b c	R.a = R.b / R.c
SUBI a b ic	R.a = R.b - ic	DIVI a b ic	R.a = R.b / ic
SUBIU a b uc	R.a = R.b - uc	DIVIU a b uc	R.a = R.b / uc
integer comparison		**integer modulo**	
CMP a b c	R.a = cmp(R.b,R.c)	MOD a b c	R.a = R.b % R.c
CMPI a b ic	R.a = cmp(R.b, ic)	MODI a b ic	R.a = R.b % ic
CMPIU a b uc	R.a = cmp(R.b, uc)	MODIU a b uc	R.a = R.b % uc
logical or		**logical xor**	
OR a b c	R.a = R.b | R.c	XOR a b c	R.a = R.b ^ R.c
ORI a b ic	R.a = R.b | ic	XORI a b ic	R.a = R.b ^ ic
ORIU a b uc	R.a = R.b | uc	XORIU a b uc	R.a = R.b ^ uc
logical and		**logical bic**	
AND a b c	R.a = R.b & R.c	BIC a b c	R.a = R.b & ~R.c
ANDI a b ic	R.a = R.b & ic	BICI a b ic	R.a = R.b & ~ ic
ANDIU a b uc	R.a = R.b & uc	BICIU a b uc	R.a = R.b & ~ uc
logical shift		**arithmetic shift**	
LSH a b c	R.a = lsh(R.b,R.c)	ASH a b c	R.a = ash(R.b,R.c)
LSHI a b ic	R.a = lsh(R.b, ic)	ASHI a b ic	R.a = ash(R.b, ic)
bound check			
CHK a c	raise an error if not 0 <= R.a < R.c		
CHKI a ic	raise an error if not 0 <= R.a < ic		
CHKIU a uc	raise an error if not 0 <= R.a < uc		

where

`cmp(b,c)` = a value with the same sign as `b - c` but with a possibly different magnitude

```
lsh(b,c) = c > 0 ? b << c : b >>> -c
ash(b,c) = c > 0 ? b << c : b >> -c
```

1.2 Load/Store Instructions

Instruction	Signification	Description
LDW a b ic	R.a = <word at address R.b + ic>	load word
LDB a b ic	R.a = <byte at address R.b + ic>	load byte
POP a b ic	R.a = <word at address R.b> R.b = R.b + ic	pop word
STW a b ic	<word at address R.b + ic> = R.a	store word
STB a b ic	<byte at address R.b + ic> = (byte)R.a	store byte
PSH a b ic	R.b = R.b - ic <word at address R.b> = R.a	push word

1.3 Control Instructions

Instruction	Signification	Description
`BEQ a oc`	`PC += 4 * (R.a == 0 ? oc : 1)`	branch if equal
`BNE a oc`	`PC += 4 * (R.a != 0 ? oc : 1)`	branch if not equal
`BLT a oc`	`PC += 4 * (R.a <  0 ? oc : 1)`	branch if less than
`BGE a oc`	`PC += 4 * (R.a >= 0 ? oc : 1)`	branch if greater or equal
`BLE a oc`	`PC += 4 * (R.a <= 0 ? oc : 1)`	branch if less or equal
`BGT a oc`	`PC += 4 * (R.a >  0 ? oc : 1)`	branch if greater than
`BSR  oc`	`R.31 = PC + 4` `PC += 4 * oc`	branch to subroutine
`JSR  lc`	`R.31 = PC + 4` `PC  = 4 * lc`	jump to subroutine
`RET   c`	`PC  = R.c`	jump to return address

1.4 Miscellaneous Instructions

Instruction	Signification	Description
`BREAK`	`<break execution>`	break execution
`SYSCALL a b uc`	`R.a = SYS_uc(R.a,R.b)`	invoke a system function

2 System Calls

♯	Instruction	Signification
1	`SYSCALL a 0 SYS_IO_RD_CHR`	`R.a` = Unicode of read character or -1 if `EOF`
2	`SYSCALL a 0 SYS_IO_RD_INT`	`R.a` = value of read integer
6	`SYSCALL a 0 SYS_IO_WR_CHR`	write character with Unicode `R.a`
7	`SYSCALL a 0 SYS_IO_WR_INT`	write signed value `R.a` in decimal format
15	`SYSCALL 0 0 SYS_IO_FLUSH`	flush the output stream
11	`SYSCALL a b SYS_GC_INIT`	initialize the garbage collector
12	`SYSCALL a b SYS_GC_ALLOC`	`R.a` = address of a newly allocated block of `R.b` bytes
13	`SYSCALL a 0 SYS_GET_TOTAL_MEM_SIZE`	`R.a` = total memory size in bytes
19	`SYSCALL a 0 SYS_EXIT`	terminates the emulation with status code `R.a`

3 Constants

Notation	Size	Signification
`a`, `b`, `c`	5 bits	Register number
`ic`	16 bits	Signed integer
`uc`	16 bits	Unsigned integer
`oc`	21 bits	Signed displacement
`lc`	26 bits	Absolute address

4 Registers

The `DLX` processor has 32 registers which are 32 bits large.
The register `R0` is always equal to 0 and is immutable.
The register `R31` is used to save the return address (`BSR` and `JSR`).
By convention, `R1` is used to store the result of a function call and `R30` is the stack pointer.