V: oVerflow. An arithmetic overflow occurred on a signed operation (8-bit or 16-bit only).

For addition information on the status bits, see section 7.1 (Sign, carry, and overflow).

To avoid potential deadlock, X-Makina enforces the following with respect to the sleep-state bit (SLP):

1. When entering or leaving an interrupt service routine, PSW.SLP is cleared.
2. When the CPU priority is 7, PSW.SLP cannot be set.

This register should not be used as a general purpose register.

4.5 Program counter (R7 or PC)

The program counter, PC, contains the address of the next instruction to be executed. Instructions must fall on even-byte boundaries, therefore the hardware always clears the least-significant bit.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0

Moving a value to the PC is equivalent to a JUMP instruction,[3] control will pass to the specified address. Note that there will be a one or two instruction delay if X-Makina is implemented as a pipeline processor.

5 Instructions

X-Makina's 33 instructions are described in this section.

5.1 Memory access

Memory access instructions allow a program to access data memory. Two registers are used, one register specifying the effective-address (EA) of a memory location to be read-from (in this case, the contents of the memory location are copied to the second register) or written-to (in this case, the contents of the second register are copied to memory location).

There are four memory access instructions.

5.1.1 Register direct and register direct with pre or post auto-increment or auto-decrement

The format of the LD (load) and ST (store) instructions is:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode					PRPO	DEC	INC	0	W/B	SRC			DST		

The register used for memory-access can be modified using:

PRPO: Pre- or post- increment or decrement of the memory register specifying the location to access. A clear value (i.e., zero) can indicate either a post-increment or post-decrement or no action, while a set value (i.e., one) indicates a pre-increment or pre-decrement action.

[3] X-Makina does not have a JUMP instruction; however, it can be emulated; see section 6. Branching, or transfer of control, is explained in section 5.3.b

DEC: Decrement the register (before or after the instruction is executed, see PRPO). A clear value indicates no decrementing, while a set value indicates decrementing.

INC: Increment the register (before or after the instruction is executed, PRPO). A clear value indicates no incrementing, while a set value indicates incrementing.

SRC: The source register (R0 through R7). The SRC register is where the data comes "from".

The SRC register in the load instruction (LD) refers to the memory location from which a data value is read and loaded-into the DST register (see below). The SRC register can be modified (pre/post and decrement/increment) to move through memory a byte or a word at a time.

The SRC register in the store instruction (SD) contains the value of the data to be written to memory (the address of which is specified by the DST register, see below).

DST: The destination register (R0 through R7). The DST register is where the data goes "to".

The DST register in the load instruction (LD) is the register to which the data value from memory is to be written (the address of which is specified by the SRC register, see above).

The DST register in the store instruction (ST) refers to the memory location to which a data value is written-to from the SRC register. The DST register can be modified (pre/post and decrement/increment) to move through memory a byte or a word at a time.

The possible address modifier combinations are listed in Table 2. A modified register used to indicate an address in a byte load or store will increment or decrement by 1, while a register referring to a word will increment or decrement by 2.

Table 2: Valid PRPO, DEC, and INC combinations and their meanings (PRPO, DEC, and INC combinations 011, 100, and 111 are undefined)

Register format	Definition	Effective address (EA) and register value	PRPO	DEC	INC
Rn	Unmodified register	*EA = Rn* *memory[EA]*	0	0	0
+Rn	Pre-increment register	*EA = Rn + 1 (byte) or* *EA = Rn + 2 (word)* *memory[EA]* *Rn = EA*	1	0	1
Rn+	Post-increment register	*EA = Rn* *memory[EA]* *Rn = Rn + 1 (byte) or* *Rn = Rn + 2 (word)*	0	0	1
-Rn	Pre-decrement the register	*EA = Rn - 1 (byte) or* *EA = Rn - 2 (word)* *memory[EA]* *Rn = EA*	1	1	0
Rn-	Post-decrement the register	*EA = Rn* *memory[EA]* *Rn = Rn - 1 (byte) or* *Rn = Rn - 2 (word)*	0	1	0

The uses of the source (SRC) and destination (DST) registers depend on the instruction (LD or ST) and are explained in Table 3.

Table 3: Load and store register-direct and register-direct with pre- or post-auto-increment or auto-decrement

Instruction	Operation	Description	Opcode
LD(.B or .W) SRC,DST **LD(.B or .W) +SRC,DST** **LD(.B or .W) -SRC,DST** **LD(.B or .W) SRC+,DST** **LD(.B or .W) SRC-,DST**	*if Pre-Incr or Pre-Decr then* *SRC = SRC + address modifiers* *EA = SRC* *DST ← memory[EA]* *if Post-Incr or Post-Decr then* *SRC = SRC + address modifiers*	Load a register (DST) from memory location specified by the effective address (EA), the value in the SRC register, modified or unmodified. Reading a byte stores the value in the low-byte of the DST register; the high-byte is unchanged.	1.0000
ST(.B or .W) SRC,DST **ST(.B or .W) SRC,+DST** **ST(.B or .W) SRC,-DST** **ST(.B or .W) SRC,DST+** **ST(.B or .W) SRC,DST-**	*if Pre-Incr or Pre-Decr then* *DST = DST + address modifiers* *EA = DST* *memory [EA] ← SRC* *if Post-Incr or Post-Decr then* *DST = DST + address modifiers*	Store a register (SRC) in memory location specified by the effective address. Writing a byte to the low-byte of a word does not change the word's high-byte.	1.0001

The LD and ST instructions permit array accessing (e.g., 8-bit and 16-bit arrays);[4] for example, to copy 10 words from Array1 to Array2:

[4] Note: An 8-bit array of characters is a string.

```
      MOVL        Array1,R2   ; Ptr1 = &Array1
      MOVH        Array1,R2
      MOVL        Array2,R3   ; Ptr2 = &Array2
      MOVH        Array2,R3
      MOVLZ       #10,R0      ; Counter = 10
Loop  LD          R2+,R1      ; Data = *Ptr1++
      ST          R1,R3+      ; *Ptr2++ = Data
      SUB         #1,R0       ; Counter = Counter - 1
      BNZ         Loop        ; If Counter ≠ 0 Then repeat from Loop
```

R2 and R3 are incremented by 2 since LD and ST are operating on words.

The number of bytes in a NUL-terminated string can be counted using LD and CMP:

```
      MOVL        String,R1   ; char *ptr = Stringb
      MOVH        String,R1
      MOVLZ       #0,R0       ; count = 0
Loop  LD.B        R1+,R2      ; data = *ptr++
      CMP.B       #0,R2       ; if data = NUL, leave loop
      BEQ         Done
      ADD         #1,R0       ; count++
      BAL         Loop
Done                          ; R0 (count) has length of string
```

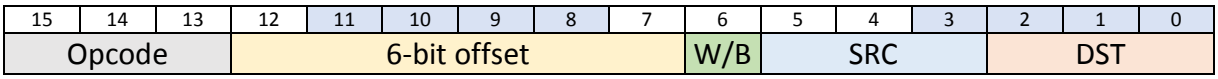
In the above example, R1 is incremented by 1 because LD.B operates on bytes

These instructions can also be used to create stacks and stack operators. For example, LD and ST can be used to implement stack instructions PUSH and PULL (POP).

```
SP    EQU   R5                ; SP equated stack pointer
; Stack is pointed to by SP
      MOVL  STKTOP,SP
      MOVH  STKTOP,SP
; Push R2 onto stack, allowing R2 to be used
      ST    R2,-SP            ; Push R2 (save R2)
      MOVLZ #0,R2             ; Clear R2
; Other instructions involving R2
      LD    SP+,R2            ; Pull R2 (restore R2)
```

5.1.2 Register relative

X-Makina also supports two load and store instructions, LDR and STR, which use register-relative addressing where the effective address is determined from the register value plus the value of a 6-bit signed offset stored in the instruction:



The 6-bit signed offset (bits 6 through 11) values are extracted and stored in bits 0 through 5 of an internal 16-bit register with the value of bit 5 being duplicated in bits 6 through 15 (i.e., the sign-bit is extended, this is referred to as *sign-extension*).[5] The signed, shifted value in the internal register is added to the SRC or DST register (SRC for load or DST for store) to become the

[5] The internal register is not accessible by software. It is internal to the CPU.