Hardware

Registers are named

$0, $1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11, $12, $FP, $SP, $PC.

and are represented by 4 bit values in the range 0 to 15

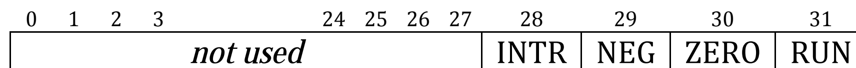
The processor flags are RUN, ZERO, NEG, INTR

conditions based on these flags are named

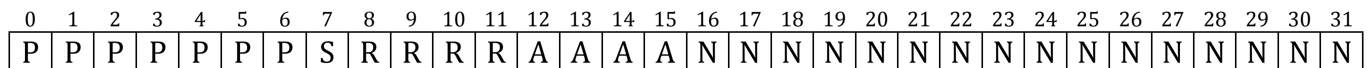
$Z	(0)	The ZERO flag is 1.
$EQ	(0)	The ZERO flag is 1.
$NZ	(1)	The ZERO flag is 0.
$NE	(1)	The ZERO flag is 0.
$LT	(2)	The NEG flag is 1, and the ZERO flag is 0.
$NEG	(2)	The NEG flag is 1, and the ZERO flag is 0.
$LE	(3)	The NEG flag is 1, or the ZERO flag is 1.
$GT	(4)	The NEG flag is 0, and the ZERO flag is 0.
$GE	(5)	The NEG flag is 0.
$POS	(5)	The NEG flag is 0.
$INTR	(6)	The INTR flag is 1.

and are represented by the four bit values shown in parentheses.

The flags register contains all of the flags in one word, as follows:



The format of a 32 bit instruction is



P represents the 7 bit operation code

S represents the star, a 1 bit indirect flag

R represents the register to be used, or in some cases the condition to be tested, 4 bits

A represents the auxilliary register in 4 bits; 0 indicates "none"

N represents the 16 bit numeric part of the operand

bit 16 (bit 0 of N) is N's sign bit. In use, N is expanded to a 32 bit value, by filling its first 16 bits with copies of bit 16. This preserves the sign for positive and negative values.

Effective address calculation, performed before each instruction is executed. The result is called OV, for Operand Value.

1. OV = N, extended to 32 bits by extending the sign bit.
2. if A is not zero, then OV += register[A]
3. if S is 1, then OV = memory[OV]

Instructions arranged in functionally similar groups (details in later sections)

data transfer

1	LOAD	register = OV
2	LOADH	most significant 16 bits of register = OV

50	LDVRZ	load value of register 0: register[R] = register[0]
3	STORE	memory = register
35	STOREH	memory = most significant 16 bits of register
4	ZERO	set memory location to zero

arithmetic

5	ADD	add OV to register
6	SUB	subtract OV from register
10	RSUB	register = OV minus register
7	MUL	register = register times OV
8	DIV	divide register by OV
11	RDIV	register = OV divided by register
9	MOD	register = register modulo OV
12	RMOD	register = OV modulo register

13	INC	increment memory[OV]
51	INCR	increment register[R]
14	DEC	decrement memory[OV]
52	DECR	decrement register[R]
31	NEG	register[R] = - OV

shifts

40	SHL	shift register left
43	SHR	shift register right
41	ASHL	arithmetic shift left, preserving sign bit
44	ASHR	arithmetic shift right, preserving sign bit
42	ROTL	rotate register left
45	ROTR	rotate register right

comparisons

15	CMP	compare register with OV, set flags accordingly
16	RCMP	compare OV with register, set flags accordingly
17	CMPZ	compare OV with 0, set flags accordingly
49	MAX	register = maximum of register and OV
48	MIN	register = minimum of register and OV

bitwise logic

27	AND	register = register & OV, bitwise and
28	OR	register = register | OV, bitwise or
30	NOT	register = ~ OV, bitwise not
29	XOR	register = register ^ OV, bitwise exclusive or

flags and conditions

34	ANDTF	and to flags: set flags according to result of register[R] & OV
32	LDFLS	load OV into flags
33	STFLS	save flags into memory

jumps

18	JUMP	unconditional jump
19	JCOND	jump if condition is true

20	JCNDF	jump if condition is false
25	CALL	push PC and jump
26	RET	pop PC

stack

22	POP	pop into register
24	POPA	pop all: registers and flags
21	PUSH	push OV
23	PUSHA	push all: flags and all registers

bit and byte manipulation

46	MKSEL	make bit or byte selector
37	GTBFR	get byte from (from simple value)
36	GTBOF	get byte of (from array)
39	PTBFR	put byte from (into array)
38	PTBOF	put byte of (into single memory location).

input and output

124	INCH	input character to memory
123	INN	input number to memory
125	OUT	output OV in detauil for debugging
121	OUTCH	output character
120	OUTN	output number
122	OUTS	output string

other

0	BAD	Error, set RUN=0
127	HALT	RUN = 0

Instructions in numeric order on opcode

0	BAD	Error, set RUN=0
1	LOAD	register[R] = OV
2	LOADH	most significant 16 bits of register[R] = OV; other 16 unmodified
3	STORE	memory[OV] = register[R]
4	ZERO	memory[OV] = 0
5	ADD	register[R] = register[R] + OV
6	SUB	register[R] = register[R] - OV
7	MUL	register[R] = register[R] * OV
8	DIV	if OV=0 then RUN=0, else register[R] = register[R] / OV
9	MOD	if OV=0 then RUN=0, else register[R] = register[R] % OV
10	RSUB	register[R] = OV - register[R]
11	RDIV	if register[R]=0 then RUN=0, else register[R] = OV / register[R]
12	RMOD	if register[R]=0 then RUN=0, else register[R] = OV / register[R]
13	INC	memory[OV] = memory[OV] + 1
14	DEC	memory[OV] = memory[OV] + 1
15	CMP	if register[R] = OV, then ZERO=1 else ZERO=0; if register[R] < OV, then NEG=1 else NEG=0;

16 RCMP if OV = register[R], then ZERO=1 else ZERO=0;
if OV < register[R], then NEG=1 else NEG=0;
17 CMPZ if OV = 0, then ZERO=1 else ZERO=0; if OV < 0, then NEG=1 else NEG=0;
18 JUMP PC = OV
19 JCOND if condition encoded in R is true, then PC = OV
20 JCNDF if condition encoded in R is false, then PC = OV
21 PUSH SP = SP – 1; memory[SP] = OV
22 POP register[R] = memory[SP]; SP = SP + 1
23 PUSHA flags then registers 1 to 12 pushed, as above
24 POPA registers 12 to 1 then flags popped, as above
25 CALL SP = SP – 1; memory[SP] = PC; PC = OV
26 RET if SP = 0 then RUN = 0, else { PC = memory[SP]; SP = SP + 1 }
27 AND register[R] = register[R] & OV, bitwise and
28 OR register[R] = register[R] | OV, bitwise or
29 XOR register[R] = register[R] ^ OV, bitwise exclusive or
30 NOT register[R] = ~ OV, bitwise not
31 NEG register[R] = - OV, numeric negation
32 LDFLS flags = OV
33 STFLS memory[OV] = flags
34 ANDTF NEG = 0; if register[R] & OV (bitwise and) = 0 then ZERO=1 else ZERO=0
35 STOREH memory[OV] = register[R] shifted right by 16 bits
36 GTBOF See bytes and selectors, below.
37 GTBFR See bytes and selectors, below.
38 PTBOF See bytes and selectors, below.
39 PTBFR See bytes and selectors, below.
40 SHL register[R] = register[R] shifted left by OV bits.
if all OV bits lost at left were 0, then ZERO=1 else ZERO=0
41 ASHL register[R] = register[R] shifted left by OV bits. Sign bit is not changed.
if all OV bits lost at left were 0, then ZERO=1 else ZERO=0
42 ROTL register[R] = register[R] shifted left by OV bits.
simultaneously, the bits lost at the left are shifted in from the right.
if all OV of those bits were 0, then ZERO=1 else ZERO=0
43 SHR register[R] = register[R] shifted right by OV bits.
if all OV bits lost at right were 0, then ZERO=1 else ZERO=0
44 ASHR register[R] = register[R] shifted right by OV bits. Sign bit is not changed.
if all OV bits lost at right were 0, then ZERO=1 else ZERO=0
45 ROTR register[R] = register[R] shifted right by OV bits.
simultaneously, the bits lost at the right are shifted in from the left.
if all OV of those bits were 0, then ZERO=1 else ZERO=0
46 MKSEL most significant 8 bits of register[R] = OV – register[R] + 1,
least significant 24 bits of register[R] = OV. See bytes and selectors.
47 *not used*
48 MIN register[R] = minimum of register[R] and OV
49 MAX register[R] = maximum of register[R] and OV
50 LDVRZ register[R] = register[0]
51 INCR register[R] = register[R] + 1
52 DECR register[R] = register[R] - 1
53 - 119 *not used*
120 OUTN OV printed in decimal
121 OUTCH character with ascii value = OV printed

122 OUTS eight bit bytes starting from most significant 8 bits of memory[OV] are printed as characters until a zero byte is encountered
123 INN memory[OV] = number entered in decimal at keyboard
124 INCH memory[OV] = ascii code of character typed at keyboard
125 OUT prints PC and OV in hexadecimal, decimal, and as characters
127 HALT RUN = 0

Instructions in alphabetic order on mnemonic

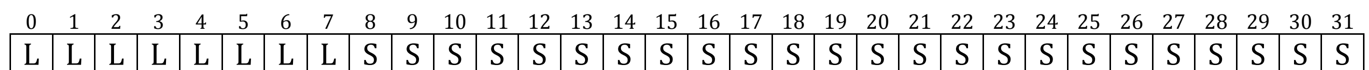
5 ADD register[R] = register[R] + OV
27 AND register[R] = register[R] & OV, bitwise and
34 ANDTF NEG = 0; if register[R] & OV (bitwise and) = 0 then ZERO=1 else ZERO=0
41 ASHL register[R] = register[R] shifted left by OV bits. Sign bit is not changed. if all OV bits lost at left were 0, then ZERO=1 else ZERO=0
44 ASHR register[R] = register[R] shifted right by OV bits. Sign bit is not changed. if all OV bits lost at right were 0, then ZERO=1 else ZERO=0
0 BAD Error, set RUN=0
25 CALL SP = SP – 1; memory[SP] = PC; PC = OV
15 CMP if register[R] = OV, then ZERO=1 else ZERO=0; if register[R] < OV, then NEG=1 else NEG=0;
17 CMPZ if OV = 0, then ZERO=1 else ZERO=0; if OV < 0, then NEG=1 else NEG=0;
14 DEC memory[OV] = memory[OV] + 1
52 DECR register[R] = register[R] - 1
8 DIV if OV=0 then RUN=0, else register[R] = register[R] / OV
37 GTBFR See bytes and selectors, below.
36 GTBOF See bytes and selectors, below.
127 HALT RUN = 0
124 INCH memory[OV] = ascii code of character typed at keyboard
123 INN memory[OV] = number entered in decimal at keyboard
13 INC memory[OV] = memory[OV] + 1
51 INCR register[R] = register[R] + 1
20 JCNDF if condition encoded in R is false, then PC = OV
19 JCOND if condition encoded in R is true, then PC = OV
18 JUMP PC = OV
32 LDFLS flags = OV
50 LDVRZ register[R] = register[0]
1 LOAD register[R] = OV
2 LOADH most significant 16 bits of register[R] = OV; other 16 unmodified
49 MAX register[R] = maximum of register[R] and OV
48 MIN register[R] = minimum of register[R] and OV
46 MKSEL most significant 8 bits of register[R] = OV – register[R] + 1, least significant 24 bits of register[R] = OV. See bytes and selectors.
9 MOD if OV=0 then RUN=0, else register[R] = register[R] % OV
7 MUL register[R] = register[R] * OV
31 NEG register[R] = - OV, numeric negation
30 NOT register[R] = ~ OV, bitwise not
28 OR register[R] = register[R] | OV, bitwise or
125 OUT prints PC and OV in hexadecimal, decimal, and as characters
121 OUTCH character with ascii value = OV printed

120	OUTN	OV printed in decimal
122	OUTS	eight bit bytes starting from most significant 8 bits of memory[OV] are printed as characters until a zero byte is encountered
22	POP	register[R] = memory[SP]; SP = SP + 1
24	POPA	registers 12 to 1 then flags popped, as above
39	PTBFR	See bytes and selectors, below.
38	PTBOF	See bytes and selectors, below.
21	PUSH	SP = SP – 1; memory[SP] = OV
23	PUSHA	flags then registers 1 to 12 pushed, as above
16	RCMP	if OV = register[R], then ZERO=1 else ZERO=0; if OV < register[R], then NEG=1 else NEG=0;
11	RDIV	if register[R]=0 then RUN=0, else register[R] = OV / register[R]
26	RET	if SP = 0 then RUN = 0, else { PC = memory[SP]; SP = SP + 1 }
12	RMOD	if register[R]=0 then RUN=0, else register[R] = OV / register[R]
42	ROTL	register[R] = register[R] shifted left by OV bits. simultaneously, the bits lost at the left are shifted in from the right. if all OV of those bits were 0, then ZERO=1 else ZERO=0
45	ROTR	register[R] = register[R] shifted right by OV bits. simultaneously, the bits lost at the right are shifted in from the left. if all OV of those bits were 0, then ZERO=1 else ZERO=0
10	RSUB	register[R] = OV - register[R]
40	SHL	register[R] = register[R] shifted left by OV bits. if all OV bits lost at left were 0, then ZERO=1 else ZERO=0
43	SHR	register[R] = register[R] shifted right by OV bits. if all OV bits lost at right were 0, then ZERO=1 else ZERO=0
33	STFLS	memory[OV] = flags
3	STORE	memory[OV] = register[R]
35	STOREH	memory[OV] = register[R] shifted right by 16 bits
6	SUB	register[R] = register[R] - OV
29	XOR	register[R] = register[R] ^ OV, bitwise exclusive or
4	ZERO	memory[OV] = 0

Bytes and Selectors

The five instructions MKSEL, GTBOF, PTBOF, GTBFR, PTBFR, are provided to simplify the use of data items smaller than a whole 32 bit word. A byte is defined to be any group of consecutive bits in memory, a byte may start in one word and end in the next.

To completely describe a particular byte within a region of memory, only two numbers are needed: the length of the byte (in bits), and its beginning position, the number of bits before its start. These are called `start` and `length`. The description of a byte that they jointly define is called a `selector`. As the length must be no more than 32 bits, a whole selector may (almost reasonably) be stored in a normal 32 bit word as follows.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
L	L	L	L	L	L	L	L	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S

The eight bits labelled L are the length, the 24 bits labelled S are the starting position. The restriction of the starting position to 24 bits means that a non-array data structure may be no more than 2^{24} bits, or 2^{17} words (that is 16,777,216 bits or 131,072 words) long. Array accesses have no such restriction.

`MKSEL r, opd` uses register[r] as the starting position, and OV as the ending position, to create a selector (start=register[r], length=OV – register[r] + 1) which is stored in register [r].

`GTBOF r, opd` takes the selector already stored in register[0] to describe a range of bits inside the 32 bit value OV. Those bits are extracted and shifted right into the least significant bits of a 32 bit word, and stored in register[r].

`PTBOF r, opd` takes the selector already stored in register[0] to describe a range of bits inside the 32 bit value memory[OV]. Those bits are replaced by the value of register[r], and the result is stored back in memory[OV].

`GTBFR r, opd` takes the selector already stored in register[0] to describe a range of bits in memory starting from memory[OV]. Those bits are extracted and shifted right into the least significant bits of a 32 bit word, and stored in register[r].

`PTBFR r, opd` takes the selector already stored in register[0] to describe a range of bits in memory starting from memory[OV]. Those bits are replaced by the value of register[r], and the result is stored back into the memory locations that it originally came from.

Examples

```
    LOAD     $7, 0x5678
    LOADH    $7, 0x1234         ; makes register[7] = 0x12345678
    LOAD     $2, 12
    MKSEL    $2, 19             ; creates a selector for 8 bits starting from bit 12
    GTBOF    $1, $7
; The overall effect is to set register[1] to 0x45

    LOAD     $1, 0x5678
    LOAD     $1, 0x1234
    LOAD     $0, 148
    MKSEL    $0, 160            ; creates a selector for 16 bits starting from bit 144
    PTBFR    $1, array
; The overall effect is to set the least significant 12 bits of memory[array+3] to 0x678
; 'array' is probably the address of an array at least 4 words long.
```