A Short Description of the ELAN assembler language for the FL-XB

The FL-XB is a machine with a word length of 27 bits, the 15 least significant of which may serve as the address part of an instruction. Standard size of memory may be considered to be 32 K, expandable in 16 K modules to 256 K. In the following the contents of memory location $i$ will be denoted by $M[i]$. Integers are represented in the "ones complement" method, giving rise to 200 representations of 250 ($0$ and $-0$). Real numbers are represented by two consecutive words: "head" and "tail". The bits may be numbered $b_{53}$ (most significant), $b_0$ (least significant).

$b_{56}$ contains the sign of the mantissa (sign of the real number) but contains a copy of $b_{26}$.

$b_{53} - b_{27} b_{25} - b_0$ form the mantissa, to be considered as an integer of capacity $b_{53} - b_{27} < 2^{140}$.

$b_{52} - b_{42}$ form the (11-bit) exponent.

$b_{53}$ is the sign of the exponent.

The representation is such that for interpretation of $M[i]$ the sign of the mantissa is $-1$, then $b_{53} - b_{42}$ will be inverted.

Therefore: inversion of a real number can be accomplished by simple inversion of both head and tail.

Standardized representation of real numbers is such that the exponent is as close to 0 as possible (without losing accuracy). In particular, integers $< 2^{140}$ are represented
with exponent zero, so reals may serve as semi-integer.
More in particular: integers \(\leq 2^{12}6\) are represented in
the tail of a real in exactly the same way as
single-length integers.

The instruction code is one-address, usually
containing a register name and a memory operand.

If we want to denote the address or memory
location instead of its contents, we may write
\[ M[x] \text{ or } (\text{the address of } x) \]
A shorthand version for (as an example): \(M[324] = \) simply \(324\).

The memory operand in an instruction may
usually take on several forms which we will discuss
later. For the time being we will denote
any memory operand by \(x\).

We will discuss the registers and instructions
pertaining to them.

There are three integer registers A, S and B.

The following set of instructions is similar for
each of these three:

\[ A := x + A \]
\[ A := x \]
\[ A := -x \]
\[ A := -A \]
\[ x := A \]
\[ x := x + A \]

\[ S := x - A \]
\[ B := x - A \]

\[ \text{PLUS} A(x) \]
[These too leave their result in both A and \(x\)]

\[ \text{MINUS} A(x) \]

The B-register is almost exclusively used as
a stack-pointer so in the ELAN-code of the system
it will be found only very rarely in instructions like
1) For the A and S registers two logical operations exist, viz.

   \[ A \text{ (or S)}' + ' x \]
   \[ A \text{ (or S)}' + '- x \] (This means that \( x \) will be inverted before the operation takes place)

2) \[ A \text{ (or S)}' \times ' x \]
   \[ A \text{ (or S)}' \times '- x \]

here \( '+' \) denotes a bit by bit addition without carry

and \( '*' \) denotes the bit by bit "logical and"
the above.

In integer multiplication and division, A and S may together serve as a double-length register (we won't go into details here). In A and S, shifting operations are possible.

Left:
- \( \text{LCA}(x) \) to register, \( \text{LUA}(x) \)
- \( \text{LCS} \) to \( \text{S} \) register, \( \text{LUS} \)
- \( \text{LCAS} \) combined registers, \( \text{LUAS} \)
- \( \text{LCSA} \) cyclically, \( \text{LUSA} \)

Right:
- \( \text{RCA} \)
- \( \text{RCS} \)
- \( \text{RCAS} \)
- \( \text{RCSA} \) clean-shifting

Normalizing instructions (not very important):
- \( \text{NORA}, \text{NORS}, \text{NORAS} \)

Multiplication and division:
- \( \text{MULS}(x), \text{MULAS}(x), \text{DIVAS}(x), \text{DIVA}(x) \)
- \( \text{TENS}, \text{TENAS} \) (fast multiplication by 10)

The F-register:
All floating operations are performed in the F-register. As we have already seen some of these may actually be operations on integers. As a matter of fact, the object code of ALGOL programs performs all arithmetic in the F-register.

The following instructions pertain to the F-register:
- \( F - x \)
- \( F \times x \)
- \( F / x \)
- \( x = x \)
- \( x = -F \)
If \( x \) is not an "immediate" operand (see later) then these instructions always select two memory locations, viz., head and tail of \( x \). \( F = M[y] \) selects \( M[y] \) and \( M[y+1] \).

A modification of this set of \( F \)-instructions exists that selects only one memory operand. They are written with \( G \) instead of \( F \).

\[
\begin{align*}
G &= x & \text{fills the tail of } F \text{ with } x, \text{ the head of } F \text{ with signconsistent zero.} \\
G &= -x \\
G &= x + x \\
G &= x - x & \text{perform the specified operations in } F \\
G &= x \times x \\
G &= x / x & \text{operand.}
\end{align*}
\]

\( x = G \) ? the contents of the tail of \( F \) are transferred to \( x \).

The instruction counter \( T \). The \( T \) register contains the address of the next instruction to be executed. The 18 least significant bits from the address, the remaining 8 bits are the so-called small (one-bit) registers.

Three of these are:

1) the condition "yes" or "no"
2) the last sign "+" or "-"
3) the interruptability of the central processor "deaf" or "hearing"

Whenever a subroutine jump occurs the current value of \( T \) (including small registers) is stored in a specified location. This is the so-called link.

The value of \( T \) may be changed by any of the following transfer of control-instructions.
JUMP (x)  "JUMP over x instructions"  
i.e., add x to T and continue 
instruction fetch at M[T];
set T equal to x and 
continue. If x is an address 
then interpretation is straightforward.
if x is however of the form 
M[y] then transfer of control 
takes place to the address to 
be found in M[y]. M[y] 
appears to contain the "link". 
remain unchanged.

Small registers however
GOTOR (x)  
restoring
T is set equal to x

SUBC (x)  "has the same effect as GOTO 
but leaves the present value of 
T on top of the stack (M[B]) 
as a future link, increasing B 
by 1."

SUBCD (x)  "like SUBC but with the added 
effect of making the machine 
"deaf" (uninterruptible)

There is a set of other SUBROUTINE 
"jumps" that leaves their link in any of memory locations  M[22] through M[23] 
but, for one exception, they are not used in our system.

Execute instructions:
DO (M[x]) executes the instruction on M[x] and 
continues with the instruction following DO; unless 
M[x] contains a transfer of control instruction.
DOS (M[x]) is like DO but as a side effect
S will be filled with M[x]
(see next page)
Operands

Operands may be of 4 types

1) Static
   Such operands are of the form M[x] and denote the contents of memory location x.

2) Immediate
   They are written as :M[x] and denote the address of memory location x. This may be abbreviated into simply x.
   As an example: A + : M[1] may (and will) be written as A + 1

3) B-modified
   Such operands have the form M[B + x] and denote the contents of the memory location the address of which is calculated by adding x to the value of B. Usually x is a small positive or negative since B is used as the top of stack pointer.

4) Dynamic

The operand may appear in any of the following forms MG, MA, MS, MC, MT, MD inside brackets, each followed by an increment that may range from -256 to +255.

In M_p, p stands for any integer from 0 to 57.
M_p addressing is used for addressing variables declared at different block heights.

MG, MA, MS [ : ]

Denote the contents of the memory location the address of which is obtained by adding i to the 18 least significant bits of F-tail, A, S respectively.

The address operator : is allowed, so

A = : MS[3] is short for A = S

in case \( b_{26} - b_{18} \) of \( S = 0 \ldots 0 \)
MC  like  M G, M A, M S, but  with  the B register. Depending on the type of instruction in which this operand occurs the value of B is increased by 1 or 2. In most cases the situation is very obvious. A few examples:

\[
\begin{align*}
\text{MC} = \text{G} : & \quad M[8] := \text{F-tail}, \quad B := B + 1 \\
F &= \text{MC}[-2] \\
F &:= M[8-2], M[8-1]; \quad B := B - 2 \\
\end{align*}
\]

GOTO R(MC[-1]) Return to the link left on top of the stack by the last subroutine-call and restore small registers as before call. Decrease B by one.

DO (MC[-1]) Fetch instruction from top of stack and execute it.

(A multiple assignment is usually represented by as many DO (MC[-1]) as there are elements in the left hand list.)

Operands of type MC do not change the value of B.

MT  like the above but with the T-register. Caters for relative addressing within pieces of code that are relocatable. In particular a lot of use is made of subroutine-calls of the type SUBC (;MT[x]) and fetch-instructions like G = MT[x].

MD  like the above however now D functions as the modififying register. In actual fact M[63] serves the role of the D-(for display) register so now an extra memory contact is necessary for MD-addressing.

M_p  Now M[p] serves as the MODIFY (or extra memory contacts). Let D point to the so-called display then the elements in this display point to the base address of the local variables of blockheight 0, 1, 2, ..., as many as are in existence. (No more than 5)
Changing context means changing the value of \( D \) (M[63]) to let it point to another display. The object-code of an ALGOL program satisfies the convention that the register always equals the display element of the current block(s) so that local variables may be referenced through MA-addressing instead of MP-addressing.

Instruction-modification
A large number of instructions may be modified by adding a "condition-following" and/or "condition-setting" variant.
1) "Condition-following"
An instruction may be preceded by "Y", "N", or "I", implying that the instruction will only be executed if the condition-register is on "yes" or "no" respectively, otherwise it will be skipped.

2) "Condition-setting"
The setting of the condition-register may be achieved by instructions ending with "P", "Z", "E", "Q". The register will be set to Yes or No depending on the arithmetic result of the instruction.
- P sets to "yes" if sign bit positive, otherwise "no"
- Z sets to "yes" if result zero (all bits equal), otherwise "no"
- E sets to "yes" if result has the same sign as the last-sign register, otherwise "no"
The last-sign register is set to the arithmetic result of all instructions that are called "condition-setting".

3) "Undisturbed"
Proceeding an A or S instruction by "U" allows to test for the result without altering the contents of A or S.
Example: let \( A = 2000 \)
then "U, A = MA, Z" will set the condition register depending on whether the contents of M[2000] are...
Upon completion of this instruction still $A = 2000$

A few shortcomings of the (short) 27 bit instruction lang:

1) Since two bits are set aside for $U, Y, N$, nothing a combination of $U$ or $N$ and $U$ is not allowed

2) $G$ instructions are coded as $U$, $F$ instructions so no actual "undisturbed $F$ instructions" are possible

3) On account of 2) no $Y$ or $N$ $G$-instructions are allowed (This is really a nuisance)

Special memory locations

As we have already seen $M[0]$ through $M[23]$ may serve as locations for links used (not used in our system) with the exception of $M[17]$ where the link of occurrence of an interrupt is stored.

$M[24]$ through $M[28]$ contain the instructions that are executed upon occurrence of several types of interrupts as if they were executed on account of a $DO$-instruction.

We only use $M[24]$. It contains the instruction $\text{SUB}\_17$ (: INTERRUPT PROGRAM), which upon execution will leave the old link (not : $M[24]$) in $M[17]$ and transfer control to the Interrupt program.

Memory locations 57 through 63 denote

- $F$-head, $F$-tail, $A, S, B, T, D$ respectively
- $M[57]$ through $M[62]$ do not exist as such, but allow us to write instructions like

\[ A = G \quad \text{(machine-coded as : $A : M[58]$)} \]

"copy tail of $F$ in $A"$

$U, S = T, P$ (coded as $U, S = M[62], P$)

"to test for the sign bit (one of the small registers) of $T$"

(Used in evaluation of Boolean expressions)

$M[64]$ through $M[255]$ are communication channels with the peripheral computer in which four consecutive words are reserved per peripheral device, allowing for a maximum of 48 devices to be hooked on the EL-X8.

The communication commands will not be discussed here. (They may be found in END/40 and END/49)
The ELAN-assembler we use is an ALGOL-program. Its input consists of punched tape containing the ELAN-code of parts of the system. Its output is punched binary tape that may load the system into a virgin machine.

The first part of the input consists of the table in which the assembler is acquainted with:
1) the symbolic names of constants and assembly parameters
2) fixed layout of part of core memory and symbolic physical addresses

Symbolic names for dynamic addresses. Most of these are of the form MO[q], some dynamic addresses of the ALGOL-translator however go as deep as M7-addressing. (local variables of procedures declared within procedures)

4) Symbolic names of virtual addresses ("invariant addresses"). They are introduced in the table between "≡" and "‡". The length of such a segment never exceeds 512 (our fixed page size).

Every piece of ≡‡ reserves one word in core for the "segment variable" to describe the actual location of the segment.

*) During the introduction of static addresses some very frequently used operations are introduced by symbolic names derived from the current static address.

So the V-operations is introduced by the name UV allowing us to write UV instead of SUBCD( : UVOP[2])

Constants between apostrophes are octal constants
After introduction of the table a small piece of "virgin input" follows. This contains the program that reads in the remainder of the system, checks parity, and finally performs transition to the "system in action". The system itself consists of pieces of consecutive text preceded by a heading, with the symbolic name of the (virtual) address where this piece of text is to be placed and the length of this piece of text.

For reasons of efficiency the translator uses a table of its own in which all system-addresses it needs are incorporated (due care has been taken that these addresses in the two tables do coincide). The two tables could be merged into one.