9 september 1964

Description of the object program, II

(A sequel to EWD101).

Program segment transition

For the moment I restrict my attention to implicit jumps to other program segments.

In contrast to previous thoughts I have the idea, that it would be nice if such a transition could be coded in such a way that no register contents will be destroyed. (This means that many of the previous remarks about "not separated by program segment transition" can be omitted in many cases.) At present I take the view that the program segment transition will be fully preserving. (Including Interruption Permits, Conditions etc., for at present I do not know how we shall code the boolean expressions.)

In the object code program segment transition will be coded by two consecutive words, stored on the segment to be left, viz.

SE 33 Program Segment Transition

invariant target address

The second word contains 18 bits for the physical address of the segment SV and 9 bits for the line number within the segment.

SE 33 will be a SUBCD instruction.

An immediate consequence is that we are not allowed to let B point to a place of interest whenever a program segment transition occurs. The contents of this place will be overwritten by the machine link. (The translator must be jolly well aware of this rule, e.g. when translating complex expressions!)

The phenomenon of program segment transition has nothing to do with the stack, with Display-modifications or what not. Secondly it cannot be nested per program. My suggestion therefore is that it shall use the bottom of the current stack. Each program will have some global variables for this and may be other purposes. The idea of the "standard locals" suggests that one of the "standard globals". Decision is left to the system makers.

SE33 is an inconditional SUBCD instruction; making it a conditional instruction would lead to disaster as the next word (invariant target address) does not allow its interpretation as a machine instruction.

The conditional forward jump resulting from the if-clause will be coded as a conditional forward JUMP instruction to a point in the same program segment. It will point either to the direct continuation of the program or to an "transition gate" of two words, the first of which is a SE33. The inconditional jumps derived from the if then else construction will also be coded primarily as a (inconditional) JUMP instruction, leading either to the continuation of the program or to an transition gate. The fact that in the latter case we have wasted one word of program space will be accepted as the price for ease of translation.

The transition gate will also be generated in the case of segment exhaustion.

Designational expressions

We shall now describe in its full pomposity and clumsiness the general mechanism for evaluation and use of designational expressions. Whether this mechanism will be used for goto statements with explicit labels in the same block is, at this of the description, immaterial.

On account of the fact that a formal parameter specified as label may occur in the value list, we must separate the evaluation of a designational expression on the one hand and its use in a goto statement on the other.

On account of the fact that a goto statement with an undefined switch element as its final operand is to be performed as a dummy state, the possible range of label values will be extended with a special one, called "skip".

The value of an designational expression is called a label value.

The constituents of a label value unequal to skip are

1. the invariant target address
2. the local reference point of the target level.

The local reference point LRP of the target level will give access to:

LRP[0] WP of target level

LRP[1] SV-link of target level

LRP[2] D value of target level

LRP[3] CBH of target level.

Remark: To have CBH also among the standard locals is nice but unnecessary, for two reasons.
The CBH of target level could be one of the constituents of the label value.
Further more, the LRP occurs once in the display identifier by D (viz. LRP = D[CBH]) so we could find CBH by scanning of the display.
Even D is not strictly necessary because the three counters attached to each display enable us to break down the stack level by level until the LRP specified is the current one.
Specifying D as one of the standard locals, however, permits us to make a short cut. The same holds for the way in which the SV-chain and the stack page chain has been built up.

Remark. Having direct access to the target D and the target CBH allows the short cut in the goto statement. The goto should consist of the following actions, as far as stack control is concerned.

From the situation about to be left, the system must isolate

a) old SV-link := M[old LRP + 1]
b) old AP.

The first one will control array segment cancelling, the second will control stack page cancelling.

Then D := M[new LRP+2] to set new D|
D[-1] := M[new LRP+3] to set CHB|
D[-3] := 0 (i.e. new PDC).

AP := M[new LRP+0]

because at inter statement situtation AP = WP.
Now the array segment cancelling will be controlled by old SV-link and new SV-link = M[new LRP+1] and stack page cancelling will be controlled by old AP and AP. It can be proved that all associated counters of the Displays which are retained in the stack will have the correct value.

Further more program segment transition has to be effected on account of the machine link and the invariant target address. This is the usual manner, as in SE 33. (In the case of a wild jump I do not expect to distinguish between a jump within the same program segment and one leading to another.) The above was a premative discussion of the goto.

Per program we shall have two consecutive words in the bottom of the stack. This global variable has the name "universal label". All evaluation of designational expression has to deliver the invariant target address and the target LRP in these two locations: they will be implemented as assignments to the universal label. The evaluation of switch element will also act as an assignment to the universal label.

The general implementation of the goto will be implemented by a single word in the object program

SE 34 General goto

it will transfer control and regulate stack collapse on account of the target specification in universal label.

When the object program has been generated, the invariant target address will look like a constant to be stored on the same program segment.

A goto to a wild, but yet explicit label, will be coded as

S := invariant target address
UL[0] := S
S := D[target BH]
UL[1] := S

SE 34 General goto

(This needs one additional word in a program segment for the invariant target address.)

Supplying an explicit label as actual parameter consists of
1. placing a two word constant on top of the stack (the first will be a system entry of the type SUBCD the second one may be needed to indicate the fact that the parameter is a constant label value).
2. followed by

S:= invariant target address

LC[0]:= S

S:= D[target BH]

UL[0]:= S

f[0] will be filled with such a system entry, that

DOS(f[0])
will perform

UL[0]:= f[2]

UL[1]:= f[3].

If a formal label occurs in the value list it will give rise to the following piece of program

DOS(f[0])

f[0,1]:= constant described above

f[2,3]:= UL[0,1]

and the label will remain to be treated as a formal label.
Conditional designational expressions present no special difficulty.

A separate description, however, is required for the switch, i.e. the switch designator and the switch declaration.

To the switch declaration corresponds a label procedure; the full procedure mechanism will be invoked. As far as embedding in the stack is concerned it will be treated as a non-type procedure, as it will not leave its value in the stack but in the Universal Label. Analogously it will be treated in the stack as a procedure without parameters, its subscript value being rendered to it in the global variable "Switch Subscript". There is no objection to using the Universal Label for this purpose.

The switch designator will be coded as follows

F:= sunscript value

UL:= F

[calling sequence of the
switch as non-type procedure
without parameters

After return UL will contain the label value.

Remark: The switch can be formal or explicit, just as in the case of the statement procedure. Initial B:= B + 4 can be omitted and SE 12 will be used as return specifier.

The description of the switch body

The switch body is translated as a non-type procedure without parameters and no normal inner blocks within the fictitious block. (On account of type procedure calls with NSIS-parameter, the fictitious block may contain inner parameter blocks.)

The body will start with Create display (SE18); the parameter needed

for the maximum inner block height implies a scan over the complete switch declaration.

Analysis of number and type of formal parameters will be omitted.

Next the body will ask for anonymous space in order to be able to evaluate subscript expressions; it will do so with SE1, the parameter needed will imply a scan over the complete switch declaration.

Then the contents of the Universal Label will be placed in the F-register. It will be inspected for integrety (and rounded to nearest integer).

Unless segment transition disallows, you code

```
        S:= -G
        S + 1 , P
    N ,JUMP (2) ⟶
        SE 35          UL:= skip
        DO(MA[-3]) ⇒
⟶      ......
```

System entry SE 35 will make UL = skip. (A system entry is the easiest way to fill in this constant.

Then the text continues with

```
        S + number of elements dealt with on this page, P
    N , JUMP(.....) ⟶
```

If not all elements can be treated on this page, the last jump will point to a transition gate to the next.
If the complete switch can be dealt with within this page it will point to

```
⟶  SE 35
    DO(MA[-3])
```

in order to create a skip for index values above the maximum

The jump itself is followed by

```
    S + constant
```

(specifying the place of the primary element of the distributor) followed by

```
    JUMP(S)
```

The end of the page will contain the distributor, a sequence of JUMP-instructions with fixed negative argument; each of these will point to a piece of program derived from the corresponding switch element. The structure of such an element will be:

```
Evaluate Designational Expression (in UL)
DO (MA[-3]) ⇒
```

if it was impossible to accommodate the complete switch in the segment in question, the transition gates to the next segment fall into distinct classes:

one transition for the elements not dealt with by the distributor of the current segment;

one or more transition gates for the continuation of the evaluation of the last element accommodated within the current distributor.

The next segment will start with the continuation of this last element; after that it will see to the remaining switch elements. Etc.

Cumulative list of system entries introduced in EWD 102

SE 33	Program segment transition
SE 34	General goto
SE 35	Assign "skip".