10.1 BRANCHED PROGRAMS

10.1.1 A program may be divided into branches which are regarded as independent programs from the point of view of the time-sharer. By setting up branches to deal with peripheral transfers for instance the efficiency of a program may be significantly increased. The branching of a program may, however, easily give rise to program mistakes and must be used with caution.

10.1.2 Setting up a branch

A branch is set up by means of a special 150 instruction

```
i.e.              150     X       Y       24
```

This instruction means:-

Set up a branch which I shall refer to as branch X and set its entry address equal to Y. If branch X already exists make it jump to Y.

X is a small integer lying between 1 and 7 but if this is the first branch to be set up then X must not be equal to 1 and in this case the original program is regarded as branch 1.

New branches are initially switched off, waiting for the branch which set them up.

10.1.3 Reservations

Each program in the machine needs a directory entry of at least 16 words and when a new branch is set up the space for this is taken from the highest available region of the programs reservations, which are accordingly reduced. In addition to this, 8 words must be allowed for the branch directory, where the monitor program keeps details of the branch relationships. (I.e. interlocks) In general, a program intending to have n branches should request at least 16n-8 additional working store locations.

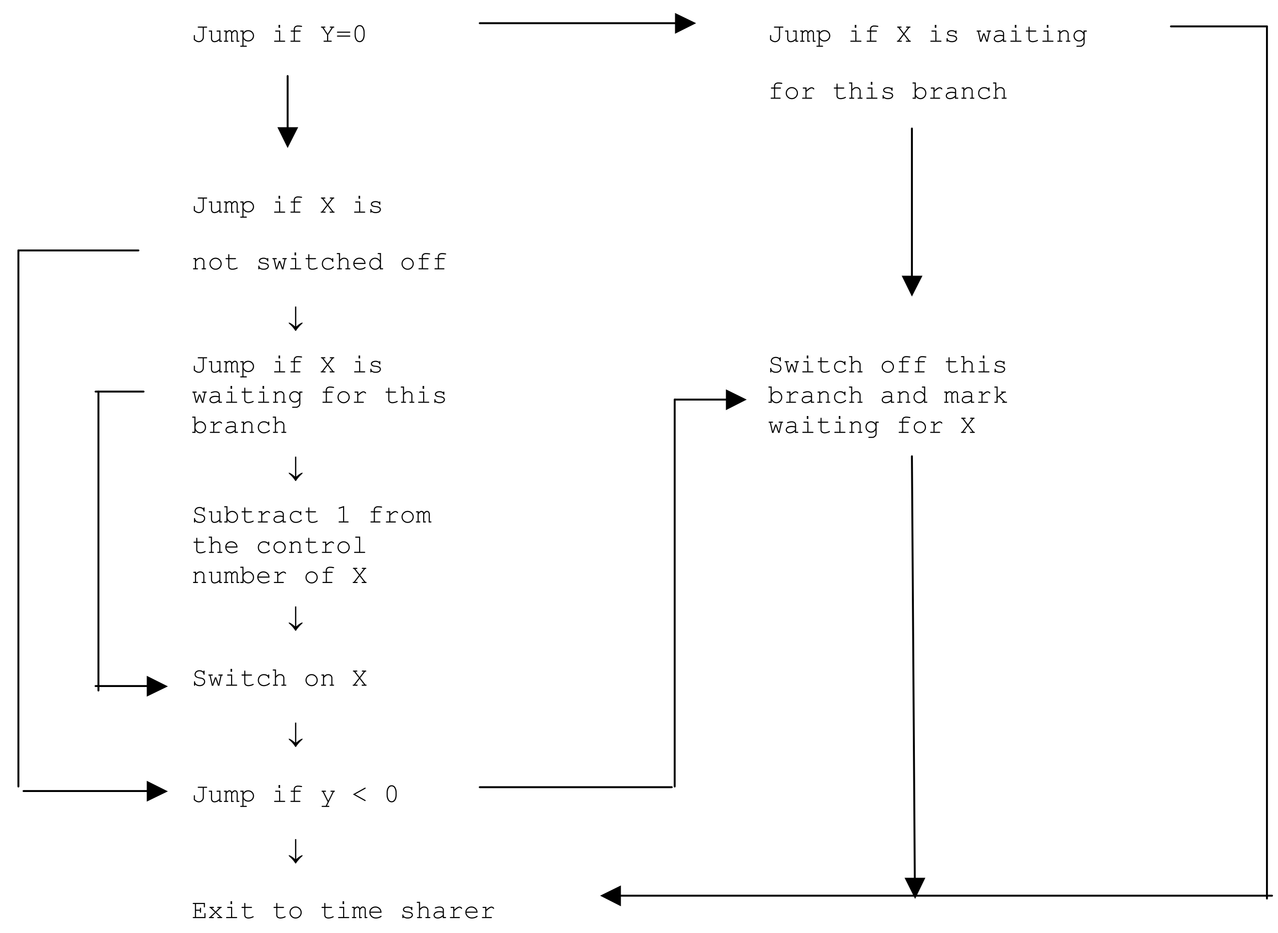
All the branches of a program are given the same core, drum and peripheral reservations, thus allowing information dumped by one branch to be picked up by another and also allowing branches to communicate by means of common markers.

10.1.4 <u>Timing</u>

The actions of the various branches are synchronised by means of the fast special 150 instruction

150 X Y 2

The precise action of this instruction is shown in the following flow diagram


Jump if Y=0
Jump if X is waiting for this branch
Jump if X is not switched off
Jump if X is waiting for this branch
Subtract 1 from the control number of X
Switch on X
Jump if y < 0
Exit to time sharer
Switch off this branch and mark waiting for X

This instruction must not be premodified and replacement would usually make it prohibitively slow.

Generally speaking the action is to switch on branch X if it is waiting and to switch off this branch if y < 0. There are two exceptions.

(i) If X is switched off waiting for another branch the effect is to switch it on but to subtract one from its control number, so that it will again obey the 150/2 which caused it to switch itself off. This allows one to deal satisfactorily with cases where several branches may alter the condition on which another branch switches itself off.

(ii) Y=0. This is a special case. The effect is to switch off this branch, waiting for X, unless X is switched off waiting for this branch. The use of this special form is illustrated in the example which follows in 10.1.6.

The normal form of the 150/2 instruction (Y≠0) should usually be followed by a guard jump, i.e.

```
150   X   Y   2
 64  -1+  Y
```

This is necessary to guard against a particular sequence of events which could cause a branch to go although its count (y) is negative (see 10.1.6).

10.1.5 Multiple buffering

A distinctive feature of branched programs is the way in which buffers are dealt with ("buffer" here simply means a region of the store where information is dumped by one branch and picked up by another). If, for instance, a branch is reading in cards it will usually have several buffers and will fill them cyclically. If there are 4 buffers each of 20 words the branch must use a modifier taking successively the values 0, 20, 40, 60, 0, 20 and so on. The branch which deals with this information must have a similar but independent modifier. In addition two counters must be kept, one indicating the number of buffers left available to the input program and the other indicating the number of buffers to be dealt with. The sum of these two counts in this example will be the number of buffers less 2, their initial values depending on the actual program.

10.1.6 Example

Suppose that a program can be conveniently divided into 3 branches which we will call for convenience, INPUT, COMPUTE and OUTPUT.

INPUT (branch 2) will read in a file from magnetic tape and at some stage detect an end of file marker.

COMPUTE (branch 1) will process this data and will use OUTPUT (branch 3) to print out the results on a line printer. We will suppose, for simplicity, that there are 4 input and 4 output buffers.

The necessary organisational instructions are as follows. (They are given more concisely in the appendix).

```
               150      2        INPUTE   24
               150      3        OUTPUTE  24
```

(Call the input and output branches 2 and 3 respectively and set their entry points equal to INPUTE and OUTPUTE).

```
                14      IC1      3
                13      IC2      1
```

(Set initial values of input counts).

```
                13      OC1      1
                14      OC2      3
```

(Set initial values of output counts).

```
      COMPUTEE) 150     2        IC2      2
                64      -1+      IC2
```

(Switch on INPUT if it is waiting. Switch this branch off if $IC2_m < 0$ i.e. there is no data to process).

Process next buffer

```
                10      IC1      1
                11      IC2      1
                10      OC1      1
                11      OC2      1
```

(Deal with counters. Note that each branch steps down the counts which appear in its 150s and steps up the associated count).

```
               150      3        OC2      2
                64      -1+      OC2
```

(Switch on OUTPUT if it is waiting. Switch off this branch if $OC2_m<0$ i.e. all the output buffers are full).

```
               Jump to COMPUTEE
      E)       150      3        0        2
                65      -1+      OC1
```

(Switch off this branch until OUTPUT switches it on and simultaneously switches itself off, i.e. wait until all output buffers have been emptied. The control number E+1 is set by the input branch).

Deal with end of file.

```
      INPUTE)   Read into next buffer.
                10      IC2      1
                11      IC1      1
               150      1        IC1      2
                64      -1+      IC1
```

(Switch on Computing branch if waiting for this one. If it is switched off waiting for OUTPUT, switch it on but subtract 1 from its control number, i.e. make it obey 150   3   OC2   2 again. Switch off this branch if $IC1_m < 0$ i.e. all input buffers are full).

```
              Jump to INPUTE if not end of file.
              150       1       0        2
               65      -1+      IC2
```

(Switch off this branch until COMPUTE switches it on and simultaneously switches itself off, i.e. wait until all the input buffers have been dealt with),

```
              150      1        E+1      24
```

(Set control number of COMPUTE. Note that it is set as E+l and not E, in case there is an interruption at this point and OUTPUT gets in and switches it on.

```
              150      1        0+       2
```

(Switch on COMPUTE and switch off INPUT, waiting for COMPUTE).

```
      OUTPUTE)  Print next buffer
               10      OC2      1
               11      OC1      1
              150      1        OC1      2
               64      -1+      OC1
```

(Switch on COMPUTE if it is waiting for this branch. If it is waiting for INPUT, switch it on but subtract 1 from its control number, i.e. make it obey 150 2 IC2 2 again. Switch off this branch if $OC1_m<0$, i.e there is nothing to output).

```
              Jump to OUTPUTE
```

When the instruction labelled COMPUTEE is first obeyed it will switch on INPUT and switch off COMPUTE, waiting for INPUT. The input branch will then be entered at INPUTE and after reading in a buffer of data it will switch on COMPUTE again. This will then process one buffer of information and switch on OUTPUT. The 150/2 instructions in this program ensure that each of the branches is switched on whenever possible, and at the same time ensure that one branch does not get ahead of the others.

The necessity for the guard jumps can be seen by considering the following sequence.

COMPUTE is interrupted when it is just about to obey the instruction 150 2 IC2 2.

INPUT is entered, proceeds until it has filled the last available buffer and switches itself off because IC1 has become negative.

COMPUTE is now entered and immediately switches on INPUT.

The instruction 150 X 0 2 means switch off this branch, waiting for X unless X is switched off waiting for this branch. By using this instruction followed by suitable conditional jumps, it is possible, as illustrated here, to shut down all but one branch and thus bring the program into a standard, known condition, when special events may satisfactorily be dealt with. Which control paths to close will obviously vary with the program and the method given here is not necessarily the best.

10.1.7 Special points about branched programs

(i) A program failure in or any OMP monitoring action for or any 150 instruction for any branch causes OMP to hold up (suspend) all branches whilst OMP carries out its action.

(ii) A 150/10 (see 5.3.10) in any branch causes all branches to be stopped in the same mode.

If the whole program has been halted e.g. after a 150/10 then it will be started by a RUN directive for the branch which caused the halting, and will be in the same state as it was in, before the halting occurred, so far as branch interlocks are concerned.

(iii) The simplest way of making a branch shut down itself is to write

```
                    150  X  0  2
```

where X is the number of the branch itself. It cannot then be switched on by any of the other branches unless they reset its control number. There is a danger when using this device of losing all the control paths.

(iv) The instruction 150/11 in any branch will immediately cause the whole program to be abolished.

(v) Each branch has its own time, new branches being allotted one minute as they are set up. Timing flags (150/1) in a branch apply to that branch alone.

(vi) Different branches may monitor the same event in different styles. It is insanitary for two branches to monitor the same event in style 7 with the same .jump address.

10.1.8 Branch names and directives

When a branch is set up it is given a job name composed of the original job name together with the branch number e.g. BLOGGS2. The original job name alone, typed with a directive will be taken to apply to branch 1 or, by implication, to all branches, e.g. BLOGGS HALT will halt all branches. The name BLOGGS1 is illegal. The directives which need a branch number are MONITOR, OUTPUTON and ANSWER and TIME. All other directives apply, implicitly, to the whole program.

A RERUN (see 5.3.10 and 5.7.2.2) directive will be accepted if a branch has obeyed 150 0 2 10 instruction which causes OMP to temporarily unbranch the program (i.e. the branch interlocks are remembered and all branches are switched off awaiting Branch 1). If it is desired to restore the branch interlocks a 150/25 may be used.

ENTER (with link) directive (see 5.3.25 and 5.7.3.4) causes OMP to "push down" and to temporarily unbranch the program and to use Branch 1's control number as the link which is stored in the accumulator, say L, and to enter the routine this being Branch 1. A 150/25 in the enter routine is used to return and restore conditions e.g. 150 L 0 25 means pull up, restore previous branch interlocks and set control number for this branch to [L]m and OVR according to [L]s and current state of OVR. Any previous pushing down and temporarily unbranched state are remembered.

ENTER (without a link) causes no "pushing down" and causes OMP to switch off all branches awaiting Branch 1 and to forgot any temporarily unbranched state and any previous "pushing down". ENTER is allowed for branch 1 only.

10.1.9 <u>Peripheral Incidents in branched programs</u>

Peripheral incidents may be set dynamically by any branch of a program, but the setting is taken to apply to the program as a whole. If the incident occurs, OMP records the program as being "pushed down", and temporarily unbranches the program. If the link is required it uses Branch 1's control number. Any previous "pushing down" and temporarily unbranched state is remembered. The routine to deal with the incident should normally end with a 150/25 instruction which will pull up and return, restoring the conditions to what they were before the incident occurred. For more details see 5.3.25.

<u>Appendix to 10.1</u>

<u>Example of branched program given in 10.1.6</u>

Four input buffers and four output buffers.

$IC1_m + 1$ = no. of buffers available to INPUT. $IC2_m + 1$ = no. of buffers awaiting processing by COMPUTE. $OC1_m + 1$ = no. of buffers awaiting to be output. $OC2_m + 1$ = no. of output buffers available to COMPUTE.

<u>COMPUTE (Branch 1)</u>

```
             150      2       INPUTE    24
             150      3       OUTPUTE   24
              14     IC1         3
              13     IC2         1
              13     OC1         1
              14     OC2         3
COMPUTEE)    150      2         IC2       2
              64     -1+        IC2
        Process next buffer
              10     IC1         1
              11     IC2         1
              10     OC1         1
              11     OC2         1
             150      3         OC2       2
              64     -1+        OC2
              75   COMPUTEE      0
      E)     150      3          0        2
              65     -1+        OC1
        Deal with end of file
```

<u>INPUT (Branch 2)</u>

```
INPUTE) Read into next buffer
  10    IC2     1
  11    IC1     1
 150     1     IC1     2
  64    -1+    IC1
Jump to INPUTE if not end of file
 150     1      0      2
  65    -1+    IC2
 150     1     E+1     24
 150     1      0+     2
```

<u>OUTPUT (Branch 3)</u>

```
OUTPUTE) Print next buffer
10       OC2       1
11       OC1       1
150       1       OC1     2
64       -1+      OC1
75     OUTPUTE     0
```

Examples of the Use of the 101 Instruction

This section contains a number of detailed examples illustrating how various types of radix conversion operations can be performed by the 101 instruction. For a specification of the 101 instruction and the notation used see section 3.10.

If a number, when converted, can be represented by-eight characters or fewer, it can usually be converted by a single 101-instruction, provided that the radices are all integers.

Example 1

If the number in A1 is a binary integer in the range $-9999999 \leq (A1)_I \leq 99999999$ $(-10^7+1 \leq (A1)_I \leq 10^8-1)$ it can be converted to characters by

```
          101      CHARS    RAD      A1
```

with the program constants

```
      RAD)      +100000000

                10,10,10,10,10,10,10,10+16
```

the +16 in the last radix-character ensuring that zero is converted as 0.

Example 2

If $(A1)_I$ represents pence of a sterling quantity in the range -£99.19.11 to £999.19.11 inclusive, then it may be converted to characters with minus-sign if negative, and with full stops between pounds, shillings and pence by

```
          101      CHARS    STERL    A1
```

with

```
      STERL)    +240000

                10,10,10+16,1+32,2,10,1+32,12
```

This will convert a zero in the 10/- column as SP; to convert it as 0 the 5th radix should be 2+16.

Note that in these two examples, y is the product of the radices (i.e. $y = \prod r_i$); this is true generally when converting an integer to eight or fewer characters. Note that it is not possible to use the 101-function unless all radices are integral, e.g. to convert ounces to quarters, pounds and ounces would require radices such as 2.8 and 1.6 which cannot be produced.

A case in which y is not the product of the radices is that of converting a binary fraction to characters.

Example 3

If, in A1, is a non-negative fraction, it may be converted for printing, to 6 decimal places preceded by 0. Thus:-

```
 51      A1       1
 00      A1       RD
101      NUM      CON      A1
```

with

```
RD) +0.00000025
CON) 31,63,63,63,63,63,63,63
2+16,1+32,10,10,10,10,10,10
```

The word CON) is $0^1 1^{47} = 1.0-2^{-47}$, i.e. it is the best single-length approximation to +1.0 attainable. (A1) is shifted down 1 place to prevent the possibility of overflow (a) on adding the rounding constant which is in RD) and (b) on dividing by $1.0-2^{-47}$. This shift is compensated by giving the first radix r_0 the value 2. This first radix forces a zero which is converted as 0 (because +16). The second radix (1) also forces a zero which is converted as full-stop (or decimal point). Thereafter, the digits proper are produced. Note that since the 0 produced by the radix - character 2+16 is treated as significant, β being set, therefore any left-hand zeros in the fraction will be converted as numeral 0 and not as SP.

If the converted form has more than 8 characters, a single 101-instruction is not sufficient.

Example 4

Assuming that A1 contains any single-length integer, convert it for output.

```
FIRST)  14      NUM      0                | Clear NUM
        101     NUM+1    CON      A1      | Convert l.s. 8 bits
        66      FIN      4                | Test OVR, if clear, FIN
        40      A1       CON              | if (A1) ≥ 10^8, divide by 10^8
        65      SEC      A1               | Jump if quotient ≥ 0
        60      SEC      A2               | Jump if remainder = 0
        81      SEC      A1               | if <0, add 1 & test if ≠
        14      NUM      30               | if =0, put minus sign in NUM
        75      ....     0                | and exit
CON)    +100000000                        | 10^8
        10,10,10,10,10,10,10,10+16
SEC)    101     NUM      CON      A1      | convert m.s. 7 digits
FIN)    75      ....     0                | and exit
```

The first 1O1-instruction produces the l.s. 8 digits in NUM+1. If the number occupies more than eight characters OVR is set, β is set and θ left clear. OVR is tested and left clear by the 66-instruction; if the number occupies 8 characters or fewer, OVR is clear, the conversion is completed by the one 101-instruction and so the 66-instruction causes a jump to the exit point. Otherwise the number is divided by 10^8. The quotient in A1 corresponds to the m.s. digits and the remainder in A2 to the l.s. 8 digits.

If this quotient is non-negative, it can be converted directly by a second 101-instruction: the 65-instructicn tests for this case. If the quotient is negative, it may be that all the l.s. 8 characters are zeros (i.e. the number is an integral multiple of 10). If so, the m.s. characters can be directly converted by a second 101-instruction. This case will be indicated by a zero remainder on division by 10, which is detected by the 60-instruction.

The 81-instruction is obeyed if the integer is negative and converts to more than 8 characters, i.e. $N \leq -10^8$. Suppose in fact, $N=-(l0^8a+b)$ where a and b are integers. Then when N is divided by 10^8 (in the 40-instruction) the quotient will be -(a+l) and the remainder 10^8-b. The digits of b have been correctly converted by the first 101-instruction and thus it is necessary to add 1 to the quotient and convert the result. In fact if, on adding 1 to the quotient, the result is zero, it follows that OVR was set only because b has eight digits and the minus sign could not be inserted into NUM+1. In such a case the 81-instruction does not cause a jump, the minus sign character is put in the l.s. end of NUM and the sequence left. Otherwise a second 101-instruction is obeyed, converting the m.s. digits and inserting the minus sign automatically.

Example 5

To convert a sterling quantity in the range

-£9,999,999,999.19.11 to £99,999,999,999.19.11 inclusive, held as pence in A1.

```
              14          NUM         0
              101         NUM+1       CON         A1
              66          FIN         4
              40          A1          CON
              65          SEC         A1
              60          SEC         A2
              10          A1          1
              61          SEC         A1
              14          NUM         30
              75          ....        0
CON)          +240000
              10,10,10+16,1+32,2,10,1+32,12
              +100000000
              10,10,10,10,10,10,10,10
SEC)          101         NUM         CON+2       A1
FIN)          75          ....        0
```

The reasoning behind this sequence of instructions is similar to that for Example 4. Note that the 81-instruction of Example 4 is replaced here by a 10-instruction and a 61-instruction. This is because, if a sterling quantity in the given range is divided by 240000 the quotient may exceed 24 bits and thus cannot be tested completely by an 81-instruction.

Example 6

Given a signed fraction P in A1 , convert it to 13 decimal places, preceded by SP0. if positive or by -0. if negative, allowing also for the case of -1.0000000000000

```
        14      A3      0                  | Clear A3

        65      BB      A1                 | Test if F ≥ 0

        66      BB      10      A1         | If < 0, test if F=-1.0

        12      A1      0                  | If ≠ -1.0, negate fraction

        115     A3      30                 | and put minus-sign in A3

BB)     32      A1      CON+4              | Multiply F by 10^5

        31      A2      CON+2              | Multiply l.s. half by 10^8, rounded

        61      CC      CON+2   A2         | Test if latter product = 10^8

        10      A1      1                  | If so, add 1 to m.s. digits

        14      A2      0                  | and clear l.s. digits

CC)     101     NUM     CON     A1         | Convert m.s. digits

        101     NUM     CON+2   A2         | Convert l.s. digits

        00      NUM     A3                 | Add sign if F negative

        75      ....    O                  | Exit

CON)    +200000

        1,2+16,1+32,10,10,10,10,10

        +100000000

        10+16,10,10,10,10,10,10,10

        +100000
```

In this method of solution the two multiplication instructions are used to transform the fraction into two integers, that in A1 representing the five m.s. decimal places and that in A2 representing the l.s. eight decimal places. The first radix, 1, in CON+1 serves to force a zero, converted as SP in which the minus sign may be set when appropriate.