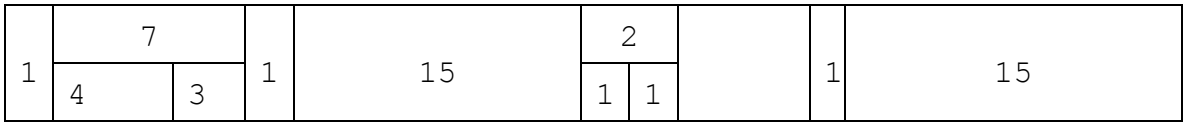
2.0 The Orion Word

In Orion the word (the content of one register or drum location) is 48 bits long. These binary digits are numbered from 0 (the most significant) to 47 (the least significant). Within the working store and on the drum a 49th bit is attached for parity checking. This parity bit is not accessible to the programmer.

The programmer may interpret words in many ways, the most common of which are described below.

2.0.1. The word as an instruction

Each machine instruction occupies the whole of one 48-bit word. The bits in the stored instruction are allocated as in the diagram and table below.



Symbol		Bits	Significance
S		D 0	Signal bit: if zero the instruction may not be obeyed: see section 5.2 page 2
F	G H	Dl to D4 D5 to D7	Function, group G (0 to 15), position in group P (0 to 7)
TX		D 8	Type (X) 1 if X-address is to be modified by z_m, or if instruction is of unmodified 2-address type
X		D9 to D23	X-address; 0 to 32767
R	RX RY	D 24 D 25	1 if X-address is to be replaced 1 if Y-address is to be replaced
Z		D26 to D31	Z-address; 0 to 63 (0 if instruction of unmodified 2-address type)
TY		D 32	Type (Y). 1 if Y-address is to be modified by z_m
Y		D33 to D47	Y-address; 0 to 32767

2.0.2 The word as a number.

Regarded as a number, a word may be interpreted in several different ways.

a) A Fixed-Point Integer

When a word is interpreted in this way, D0 is a sign bit; 0 if the number is non negative and 1 if the number is negative. Integers are held exactly: a single 48-bit word can represent any integer in the range:

$$-2^{47} \leq x_I \leq 2^{47}-1 = 140,737,488,355,327$$
$$\approx 1.4 \times 10^{14}$$

A negative integer is represented by its complement with respect to 2^{48}.

b) A Fixed-Point Fraction

When a number is regarded as a fraction, the sign of the fraction is given by the m.s. bit, in the same way as with integers. A single-length word can be regarded as a "fraction" within the range

$$-1.0 \leq x_F \leq 1.0 - 2^{-47}$$

its value being an integral multiple of 2^{-47}. (Note also that $x_F = \varepsilon x_I$, where $\varepsilon = 2^{-47}$). A negative fraction is represented by its complement with respect to 2.

If it is required to store, single length, a fraction which is not an exact multiple of 2^{-47}, the number which is actually stored is the required fraction rounded to the nearest multiple of 2^{-47}, or to the numerically greater multiple of 2^{-47} if the required number is an odd multiple of 2^{-48}.

c) A Fixed Point mixed number

Since the binary point has significance only to the programmer, and not to the computer, it follows that the programmer can imagine the binary point to be in any desired position. In general, therefore, the binary point can be considered to be n places up from the l.s. end, i.e. between D(47-n) and D(48-n). Then, with D0 as the sign bit, the (47-n) bits Dl to D(47-n) may be used to represent an integral part, and the n bits D(48-n) to D47 may be used to represent a fractional part, permitting storage of mixed numbers such as +10.7, -1257.813 etc. Denoting the value of a word regarded as a mixed number, by x_s it can be expressed as

$$x_s = x_I \, 2^{-n} = x_F . 2^{(47-n)}$$

It is an integral multiple of 2^{-n} and lies within the range

$$-2^{(47-n)} \leq x_s \leq 2^{(47-n)} - 2^{-n}$$

(As an alternative to imagining the number as being stored with the binary point somewhere along the word, it can be regarded as being stored as the integer or fraction which represents the mixed number multiplied by a suitable scaling factor; thus

$$x_I = x_s.2^n \quad \text{and} \quad x_F = x_s.2^{-(47-n)}$$

A particular case of the mixed number is the single-length mid-point number with n = 24. (2^{23} = 8,388,608).

d) A Floating-Point Number

To represent a number in standard floating-point form in Orion, the 48-bit word is divided into two fields. The first field, D0 to D39, represents the signed fractional argument of the number, and bits D40 to D47 represent the non-negative "characteristic". The signed integer equal to the characteristic minus 128 is the exponent of the floating-point number; the complete floating-point number has the value

$$x_G = x_a.2^{x_e}$$

The argument is zero or lies within one of the ranges

$$½ \leq x_a < 1 \quad \text{or} \quad -1 \leq x_a < -½,$$

depending on its sign, and the exponent lies within the range

$$-128 \leq x_e \leq 127$$

Floating-point numbers have about 11 or 12 significant decimal digits; the largest numerical value representable in this way has a magnitude of about 10^{38}.

e) Double-Length Numbers

If the contents of two consecutive registers are regarded as parts of a single number, that number is known as a double-length number. (In general, the full product of two single-length numbers is a double-length number.)

A convention normally adopted is that the m.s. bit of the second (l.s.) word is zero, the d.l. number is then said to be in 'standard form'. Regarded as a single-length number, the l.s. word is therefore non-negative. As with single-length numbers, the binary point may be imagined to be in any desired position.

The instruction with function number 126 is specifically designed for use with double-length numbers.

i) Double-length integers.

Let the values of the m.s. and l.s. words, regarded as single-length integers, be x_I and x_I* respectively. Then the value of the double-length integer formed from x_I and. x_I* is

$$x:_I = x_I.2^{47} + x_I*$$

and lies within the range

$$-2^{94} \leq x:_I \leq 2^{94}-1 \quad [2^{94} \approx 1.98 \times 10^{28}]$$

if it is in standard form (i.e. with x_I* non-negative).

The principal advantage of using double-length integers lies in the increase in the permissible magnitude of the number which can be stored.

ii) Double-length fractions

If x_F and x_F* are the values of the individual words regarded as single length fractions, the value of the double-length fraction is

$$x:_F = x_F + \varepsilon x_F*$$

where $\varepsilon = 2^{-47}$

If it is in standard form, a double-length fraction lies within the range $-1.0 \leq x:_F \leq 1.0 - 2^{-94}$

and is an integral multiple of 2^{-94}

$$x:_F = x:_I.2^{-94}$$

Use of double-length fractions increases the precision to which a general fraction can be represented in the computer.

iii) Double-length mixed number

By considering the binary point to be at some arbitrary position in the double-length number, mixed numbers can be held in double-length form.

Let the binary point be m places up from the extreme right hand end of the double-length number.

Then if $m \leq 47$, the value x_s of the double-length mixed number is $x:_s = x:_I.2^{-m}$

It lies within the range

$$-2^{(94-m)} \leq x:_s \leq 2^{(94-m)} - 2^{-m}$$

and is an integral multiple of 2^{-m}.

If, on the other hand, m >47, the value of $x:_S$ is

$$x:_S = x:_I.2^{-(m-1)}$$

and lies within the range

$$-2^{95-m} \leq x:_S \leq 2^{95-m} - 2^{-(m-1)}$$

The difference between the cases m ≤ 47 and m > 47 arises from the special treatment of the sign-bit of the second (l.s.) word in double-length working.

A particular case is when m = 47 or 48 (these two cases are equivalent, from the discussion above). The binary point is then, effectively, between the two words; the m.s. word is then regarded as an integer and the l.s. word as a (usually non-negative) fraction.

In this form, known as standard double-length mid-point representation, the number has the value

$$x:_M = x_I + x_F*$$

The range of values is $-2^{47} \leq x:_M \leq 2^{47} - 2^{-47}$

and the stored number is an exact multiple of 2^{-47}.

Double-length mid-point numbers arise naturally as the full d.l. product (produced in Orion by the 32-function) of an integer and a fraction. Double-length mid-point numbers also arise as the quotient given by the 42-instruction.

(iv) Double-length floating-point numbers.

If two consecutive words are used to store the standardized fractional argument, and a third word is used to store the exponent, then floating-point numbers can be stored with greater precision and with a larger range of exponents than can be attained with the standard packed (single-word) floating-point form.

The double-length argument is standardised to be zero or to lie, usually, within one of the ranges

$$½ \leq x:_F < 1 \text{ or } -1 \leq x:_F < -½$$

or in special circumstances

$$-½ \leq x:_F < ½$$

The instruction with function number 125 is specifically designed for standardising numbers in this form.

f) Multiple-length numbers.

Any of the number types (integer, fraction, mixed or floating-point) can be stored in more than two words if desired, to increase the possible magnitude and/or precision of the stored number.

The extensions from the double-length form to the general multiple-length form are straightforward and not detailed here (in general only the m.s. word of such a fixed-point number is allowed to be negative).

2.0.3 The word as a set of characters

Generally a character is a numeral, a letter of the alphabet, a symbol (such as + £ / : etc.) or a non-printing character associated with printing layout (space, newline, tabulate, etc.)

The 'character' which is stored within Orion is, in fact, an arrangement of bits used to represent that character inside the computer according to some arbitrary code.

One particular code, which is recommended for use whenever practicable, is the Ferranti Flexowriter Code (see Section 0.5).

Normally the internal code is such that each character is represented by a single 6-bit field; eight such fields (characters) can be packed into each 48-bit word. The eight characters in a word are conventionally denoted C0, C1, C2....C7, where C0 is that field comprising bits D0 to D5, C1 occupies D6 to D11..... and C7 occupies D42 to D47.

An alternative notation is to denote the eight characters in the word in register X by x_0, x_1, ..., x_7.

The 6-bit field representing a particular character can be regarded as a 6-bit unsigned binary integer; the value of this integer (0 to 63) is termed the value of the character.

To facilitate the input of characters to Orion, the symbolic input routine allows a line (which may or may not be labelled) on a program sheet to be written in a form typified by 0, 7, 47, 6, 50, 41, 47, 46 i.e. as eight integers in the range 0 to 63 each separated from its neighbours by a comma. Each of the successive 6-bit fields in the word is set on input to the 6-bit representation of the integer in the corresponding position of the line. Thus in the above case, the settings will be as follows:

D0 to D5;	000000	(0)
D6 to Dll;	000111	(7)
D12 to D17;	101111	(47)
D18 to D23;	000110	(6) etc.

In the standard code, those 6-bit fields represent respectively the characters SP, UC, O, LC, R, I, O, N; in effect the computer word represents the English word Orion.

In particular, if this computer word is sent to a 7-track paper-tape punch, and the punched tape is then printed via a Flexowriter, the printing appears as O r i o n , the first letter being preceded by one space.

In certain cases, notably when reading data from punched cards, a computer word is regarded as consisting of eight 6-bit fields, but these fields are not character-representations as described above. Nevertheless it is convenient and conventional to refer to such fields as characters.

Radix-words, associated with the 100- and 101-instructions, (see Section 3.10) are also regarded as being formed of eight 6-bit fields, each of which is sometimes termed a character. When it forms part of a radix-word, each character is sub-divided into two smaller fields. The second of these sub-fields, represented by the l.s. 4 bits of the six, is used as a number (0 to 15) in the arithmetical operations of the conversion process. The 2 m.s. bits which constitute the other sub-field are used to control checking (in the 100- instruction) or to determine the treatment of non-significant zeros (101-instruction).

2.0.4 The word as a set of packed data

It is often necessary to store, and possibly operate on, a number or some other item of data which can be represented by comparatively few binary digits. Useful economies in storage requirements can be achieved by packing several items of this type into a single word: there are no restrictions on the length of the fields or on the number of fields within a word (subject, obviously, to the limit imposed by the 48-bit word length).

If a field k bits long is used to hold a number, the value of that number (regarded as an integer) lies within one or other of the following ranges:-

a) if unsigned, $0 \leq n \leq 2^k - 1$

or b) if signed, with the m.s. bit of the field used as the sign bit, $-2^{k-1} \leq n \leq 2^{k-1} - 1$

Thus a 7-bit field can represent any unsigned integer in the range $0 \leq n \leq 127$ or any signed integer in the range $-64 \leq n \leq 63$.

At the programmer's discretion, packed numbers can be interpreted as integers (as above), as mixed numbers or as fractions; in the two latter cases the precision is dictated by the number of bits allocated to the fractional part. It is expected that packed numbers will normally be regarded as integers.

As stated in section 2.0.3 above, the symbolic input routine recognises lines on the program sheet such as 0, 7, 47, 6, 50, 41, 47, 46 setting each of the successive fields to the 6-bit representation of the written integers. In fact, each character can be written in other forms. For example, to set a 6-bit field to the binary equivalent of 47, any of the following written forms may be used:

a) 47

b) 32 + 15 (or the sum or difference of any number of integers, basic addresses and/or symbolic addresses, provided that the result is of rank 0 and has the required numerical value, which must be less than 64).

c) -17 (47 = 64-17, and 2^6 = 64: the l.s. 6 bits of the binary representation of the (negative) integer are stored).

The Symbolic Input Routine also accepts forms analogous to the above, to permit setting

a) four 12-bit fields, typically 127,4095,-7,100+373

b) two 24-bit fields, typically 13955, A572.

Note that, in format (b) of these, either or both of the packed quantities may be a working store address, a drum address, or the name (geographical or programmer's) of a peripheral device.

If the word is to be divided into fields of different lengths, the PACKED NUMBERS Directive (q.v.) of Symbolic Input may be used.

Naturally, the data represented by a given field can be non-numerical; the actual significance is assigned by the programmer. The use of one bit to record the sex of a person is an example of non-numerical packed data.

2.0.5 The word as a logical quantity

When a word is used as a logical quantity, interest attaches principally to the actual configuration of 0-bits and 1-bits within the word. It is usually irrelevant to consider the numerical value of the word, or the characters represented by the 6-bit fields. (Cases do arise when it is convenient to refer to and treat a number or a set of characters as though it were a logical quantity, e.g. when using instruction 123, or when using instruction 56 or 57 to copy two numbers by a single instruction.)

In particular, the m.s. bit of a logical quantity is not regarded as a sign bit, and does not receive the special treatment normally accorded to the sign bit of a numerical quantity (e.g. in the shift instructions).

One of the most common uses of a logical quantity is as a 'mask' as an operand in one of the logical instructions.

The most direct way to set, on input, the content of a word as a logical quantity is to use the MASK directive (q.v,) of Symbolic Input. Using this directive a mask consisting of ten 1-bits followed by twelve 0-bits followed by twenty-six 1 bits could be set on input by the following lines on the program sheet.

```
MASK   1-10, 0-12, 1-26

NORMAL Such a word may be labelled if desired.
```

Other means by which a logical quantity may be set on input are by writing it as:-

i) a pseudo-instruction,

ii) a number,

iii) a set of characters.

The contents of two consecutive registers may be regarded as combined to form a double-length logical quantity. The m.s. bits of the two words are not regarded as sign bits; in particular, the m.s. bit of the l.s. word is not necessarily zero, i.e. there is no "standard form" as there is with double-length numerical quantities. The instructions 56, 57 and 123 are designed specifically for operations with these double-length logical quantities.

2.1 The Working Store

2.1.1 The Working store, or core store, is a ferrite-core matrix with a capacity of between 8192 and 32768 words. Its cycle time is 12 microsecs, which determines the maximum average rate of use of the store. It is made up of registers, each holding one word of 48 bits. A 49th bit is attached to each register to provide a parity-check; the parity of each word is calculated and stored with the word in the store and is automatically checked whenever the word is used as an operand or whenever it is overwritten, unless this overwriting is caused by a 143-instruction or a peripheral reading transfer. The parity-bit is not accessible to instructions. The registers making up the working store have machine addresses which are the integers from 0 to M-1, where M is the capacity of the store. The register with machine address 0 always contains zero; the other registers can be used to hold any word. Information in the working store is volatile: it disappears if the computer is switched off.

2.1.2 Each program (or, strictly, job) has its own part of the working store, called its reserved region. This region is made up of consecutively-numbered registers starting at a register whose machine address is called the datum-point of the job. This datum-point must be a multiple of 64 and is allocated to the job by the Monitor Program immediately prior to input. Within the reserved region the addresses of the registers are denoted by A0, A1, A2, etc., which are called basic addresses. The machine address of a register can be found from its basic address by adding the datum-point to the numerical part of the basic address. (The letter A which introduces a basic address can be regarded as indicating that the datum-point has to be added - it can also be thought of as indicating an Address, since machine addresses are not normally considered.) The actual register denoted by a particular basic address will depend on the job (or program) using it.

2.1.3 The registers with basic addresses A0, A1, ..., A63 are called the accumulators of the job or program under discussion. Registers which are not accumulators (but which are within the reserved region) are sometimes called ordinary registers. The accumulator A0 always contains zero, and this register may not be written into (an attempt to do so is treated as a reservation-violation). (A0 is in fact cleared by the Monitor Program before the object program is entered.) The other accumulators are often used as working space for the job, the ordinary registers are normally used to hold program (starting traditionally at A64) and data and for extended working space - but these roles can be interchanged if necessary because the only distinction between accumulators and ordinary registers is that Z-addresses in instructions cannot refer to ordinary registers. It is thus quite legitimate to obey instructions from the accumulators.

Orion 2. The information in this section refers in general both to Orion 1 and Orion 2. Described here are the differences to be noted when reading these sections for Orion 2.

Section, 2.1.1. Orion 2 has a minimum of 16 K core-store and cycle time is 2 microseconds. Parity is checked with 143 instruction and peripheral transfers.

2.2 Instructions, general

2.2.1 An Orion instruction is made up of a number of parts, the most important being its function (F) and three addresses denoted by X, Y and Z respectively. In a written instruction these four parts are written on one line of a program sheet, on which four columns are ruled to correspond. The way in which the instruction is written depends on which program-input routine is to be used to read it in; we use here the notation of Basic Input (see Sec. 7.1) in its simplest form, which is also applicable to some other routines. The instructions making up a program are eventually punched into paper tape or cards according to conventions (punching rules) appropriate to the input routine to be used. This input routine can then be used to read in the instructions, convert them to their internal form (machine-instructions) as needed by the computer and store them. A machine-instruction occupies one 48-bit word: an instruction word.

2.2.2 A simple instruction might be written as follows:

00 A100 A293 A18

In this instruction 00 is the function, which is written in a simple numerical code, and the other three items are the X, Y and Z addresses respectively. We write:

F=00, X=A100, Y=A293, Z=A18.

The effect of this instruction is to add the content of (i.e. the number in) register A100 to the content of A293 and to place the result in register A18. The original content of A18 is lost but the contents of A100 and A293 are unchanged. This is in fact a straightforward 3-address instruction. We can indicate symbolically the effect of this instruction by writing

$$z' = x + y.$$

In this notation x, y and z represent respectively the contents of X, Y and Z and the prime (in z' on the left-hand side) indicates a reference to the content after the instruction has been obeyed. It is understood that

$$x' = x \text{ and } y' = y$$

in a general 3-address 00-instruction (provided, of course, that Z is not equal to X or Y).

2.2.3 In most instructions the X and Y addresses can refer to any of the registers reserved for the program; they are called the main addresses. The Z-address in an instruction, however, must be one of the program's accumulators (i.e. the registers whose basic addresses are A0 to A63). The datum-point of the job is normally added to the X and Y addresses (if appropriate) by the program-input routine at the time when the program is read in, these two addresses are then machine addresses; they occupy 15 bits

each in the 48-bit machine instruction. The Z-address is, however, stored without its datum-point (which is added by hardware when the instruction is obeyed) and occupies 6 bits only.

2.2.4 The function is that part of an instruction which indicates the kind of operation to be carried out; it is represented by 7 bits in the machine-instruction. The first four of these function-bits represent an integer between 0 and 15 which is the group to which the function belongs; the least-significant three function-bits represent an integer between 0 and 7: this is the position of the function within the group. The group and position numbers are written next to one another in the standard written form of the function. The following are examples:

Written function	Group-number	Position number	Function bits
02	0	2	0000 010
57	5	7	0101 111
95	9	5	1001 101
124	12	4	1100 100

The function in a written instruction is thus a 2- or 3-digit number in a mixed decimal-octal notation.

2.2.5 There are 16 groups of instructions in the instruction-repertory, each containing up to eight instructions. These groups are as follows: detailed descriptions of the instructions will be found in Sec. 3.

Group	General Description of Functions
0	y an operand) Addition, subtraction,
1	Y an operand) copying and simple
2	Pseudo-register operand) logical operations.
3	Multiplication
4	Division
5	Shifts
6,7,8	Jumps, discriminations or tests, counting
9	Floating-point arithmetical operations
10	Data-conversions
11,12	Miscellaneous logical operations
13	Spare
14	Peripheral devices, blocks of registers
15	Special operations involving the Monitor Program

The 48 bits in a machine-instruction are allocated in the way described in Sec. 2.0.1.

2.2.6 We described above (Sec. 2.2.2) the action of the 3-address 00-instruction:

```
00      A100      A293      A18
```

Most instructions also have a 2-address form. The following is a written 2-address 00-instruction:

```
00      A100      A293
```

Its effect is to add the contents of A100 and A293 (exactly as in the 3-address instruction); this result is then placed in A100, in place of the first operand. The effect of a general 2-address 00-instruction may be described as:

$$x' = x+y.$$

It will be noticed that Z is not used in the above instruction, which is described as being of "unmodified 2-address type".

2.2.7 In a general (modified) 2-address instruction the Z-address is used to specify a modifier which gets added into the X- or Y-address, or both, before the instruction is obeyed. The modifier-part of a word is simply its 24-bit less-significant half; it is denoted by a suffix m so that the modifier in A40, for example, is denoted by $A40_m$ and the modifier in z is denoted by z_m. In the instruction written

```
00Y     A100      A371      A40
```

the modifier in A40 will be used to modify the Y-address A371 (because Y is written after the function). If, for example, the content of A40 is the integer 13 at the time when the above instruction is obeyed, then this instruction will have the same effect as if it had been written

```
00      A100      A384
```

The modification process which increases the Y-address takes place in the control circuits of the computer at the instant when the instruction is obeyed; the stored instruction is not itself changed by modification. To modify the X-address in a 2-address instruction we write X after the function instead of Y. To modify both main addresses (by the same modifier) we write XY.

Any of the accumulators A1 to A63 can be used directly to hold modifiers for a program. If A0 is used we get an unmodified instruction.

2.2.8 In the stored form of an instruction the two Type Bits (TX and TY) are used to specify the main addresses which are to be modified. If either, or both, of these bits is a 1 then the corresponding main address is modified and the instruction is of 2-address type. If both of these bits are zero then the instruction is of 3-address type. Conversion from the written form of an instruction to the internal machine-instruction is done by the program-input routine used to read in the program. Conventionally it is arranged that an instruction such as

```
00      A100      A384
```

(in which only two-addresses are written and neither X nor Y is written after the function) is stored as if it had been written

```
00X     A100      A384      A0
```

The effect of the Type Bits can be summarized as follows:

(a) Both type bits zero: 3-address type
(b) Either or both type bits one, Z=0: Unmodified 2-addr type
(c) Either or both type bits one, Z≠0: Modified 2-address type

2.2.9 The Replacement Bits (RX and RY) provide the facility of indirect addressing. Either bit in a machine instruction can be made 1 by writing the corresponding main address in parentheses in the written instruction; otherwise the replacement bits are 0. Thus the instruction written

```
00      (A527)    A491      A61
```

will be stored with RX=1 and RY=O. The effect is to cause the X-address to be replaced by the modifier part of the specified register before the instruction is obeyed. If, for example, the content of A527 is the address A5OO (i.e. the integer whose value is 500 plus the datum-point) then the above instruction will have the same effect as the instruction written

```
00      A500      A491      A61
```

Replacement occurs, like modification, only in the control circuits of the computer at the instant when the instruction is obeyed; the stored instruction does not get altered.

2.2.10 Replacement can be used on either of the two main addresses (but never on the Z-address) in any instruction of whatever type. If an address is specified as being both replaced and modified then the replacement occurs first and the replaced address is then modified. For example, the instruction

```
00Y     A100      (A412)    A55
```

would have the same effect as the instruction written

```
00      A100      A319
```

if A412 contained the address A300 and A55 contained 19 at the time the instruction is obeyed.

2.2.11 In describing the effects of an instruction we use the symbols X, Y, x, y, x', y'. These refer always to the effective addresses (and their contents) - i.e. to the addresses after any modification and replacement have been carried out.

2.2.12 The modification and replacement facilities which have just been described are capable of handling most of the address-computations needed. More complicated processes can be obtained, if necessary, by using the 116- and 117-instructions described in Sec.3.11. If even these are insufficient then addresses can be computed by using any sequence of instructions and left in a register for use by means of replacement.

2.2.13 It should be noted that replacement and modification use the modifier-parts of registers, i.e. the 24-bit less-significant halves. In a machine instruction the main addresses occupy 15 bits each; these are extended to 24 bits by adjoining 9 zeros at the m.s. end when the instruction is obeyed. These 24-bit addresses are modified and replaced as necessary. The effective addresses X and Y are thus 24 bits long, in general. When such an address is used (as it frequently is) to refer to a register, only the l.s. 15 bits are employed - the m.s. 9 bits are then disregarded. The Y-addresses in shift-instructions and in 102-, 103-, 120-, 121-, 124- and 141-instructions are specially treated (see Sections 3.5, 3.10, 3.12 and 3.14).

2.2.14 Most of the instructions in the repertory have both 2-address and 3-address forms. The general rule for deriving the 2-address form from the 3-address form is:

replace z' by x' (and z*' by x*'),
replace z by zero

in the defining equations (and relations) of the 3-address form. There are, however, some instructions which do not follow this rule -notably the jump instructions of group 8.

Some instructions use or alter adjacent registers to those named. To describe these we use the notation exemplified by the following:

(a) x* is the content of X+1,
(b) z*′ is the content of Z+1 after obeying the instruction

This notation is useful, for example, in describing instructions with double-length operands, where x is the m.s. half and *x** the l.s. half of the operand. Such an operand is often written as x: (see Sec. 2.0.2e). The notation used in the Manual is summarized for reference in Sec. 0.4.

2.2.15 Signals are instructions which are marked in a certain way with a view to possible monitoring action during program-testing. They are written with a capital letter S after the function (and modification letters X or Y). For example:

01XS A1234 A9 A5

The corresponding machine-instruction has its signal-bit (see Sec. 2.0.1) equal to zero. See Sec. 5.2.1 for a description of events when a signal-instruction is encountered.

2.2.16 In the normal or "automatic" condition the instructions in a program are obeyed sequentially from the registers of the working store. The machine-address of the instruction currently being obeyed is called the control number and is sometimes denoted by c. This number gets 1 added to it during the execution of an instruction and is then available to select the next instruction. This regular sequential selection of instructions may be interrupted by a successful jump instruction, which sets c to a new value, or by an interruption connected with monitoring or time-sharing, when the current value of c is stored for possible later return. The control number is displayed on the main control panel in the lower half of J (see Sec. 2.4.4 and 2.4.5).

2.2.17 Various notations will be found in the descriptions (in Sec. 3) of the effects of various instructions. For example, an operand in X may be referred to as x, x_I , x_F , x_G , x_L or as a word made up of 6-bit characters, and so on. These notations are chosen so as to make the descriptions of the instructions as simple as possible and to indicate their most common or intended applications. They are not to be taken as indicating any restrictions on the significances of the words used with these instructions. These significances are the concern of the programmer and not of the machine.

2.2.18 There is a single overflow-indicator, commonly referred to as OVR, which is used to indicate whether capacity has been exceeded in arithmetical operations. If, for example, the words in A100 and A200 each have the value +0.75 on the fractional convention (see 2.0.2b) then the instruction

00 A100 A200 A3

will attempt to produce +1.5, which is not representable on the fractional convention. We say that capacity has been exceeded or that overflow has occurred. If monitoring on overflow is not turned on (see Sec. 5.2.3) then the overflow-indicator will be set (whether or not it was set before) and a result will be stored which is arithmetically wrong (the above instruction will place -0.5 in A3 since 0-group instructions always produce the in-range fraction congruent modulo 2 to the correct result). If monitoring on overflow is turned on then the overflow-indicator is unaltered, the instruction is not completed (i.e. no result is written away) and the Monitor Program is called in (see Sec. 5.2.3),whether or not the overflow-indicator was previously set.

The attempted computation of a floating-point number which exceeds capacity is called floating-point overflow (see Section 3.9); this will also set the overflow-indicator unless monitoring action is called for.

The state of the overflow-indicator can be found by referring to pseudo-registers 4 to 7 (see Section 2.6).

2.2.19 Overflow can occur in any instruction whose function is in this list:

00,01,02,03; 10,11,12; 20,21,22,23; 30,31,32,33,34;
40,41,42,43,44,45; 50,51,54,55; 80,81,82,83,87;
90,91,92,93,94,95; 100,101,102,103; 126.

In addition the special 150-instructions with Z = 23,30,34,52 or 53 (see Section 5.3) and the 152-instruction (not in Programmers' Mode) may cause OVR to get set.

2.2.20 Floating-point overflow can occur in any instruction whose function is:

90, 91, 92, 93, 94, 95 or 102.

2.2.21 The overflow-indicator may be cleared by instructions having the following functions:

86, 102, 125, 126

or by any instruction referring to pseudo-registers 4 or 5 (the functions of such instructions may be: 20 to 27, 66, 67, 112). (The pre-150 instruction (see section 3.15) will also clear OVR). These instructions are the only ones which clear the overflow-indicator, which will otherwise remain set after overflow has occurred.

2.2.22 Subject to the special rules governing the construction of compound instructions, which are described in Section 0.3 (Glossary, under "compound instruction"), and the special 150-instruction with Z=50, which is always followed by another word (see Section 5.3.50), any sequence or combination of legal instructions is permitted except an instruction which overwrites the instruction which is to be obeyed next. The need for tampering with stored instructions in this way is not likely to arise in practice owing to the ample provisions for modifying and replacing the addresses in obeyed instructions (see Sections 2.2.7 to 2.2.13). In those rare cases when it is necessary, e.g. when the function-part of an instruction has to be computed, then at least one instruction (possibly a dummy) must be obeyed between (a) the instruction which creates the new instruction (and overwrites the old) and (b) the new instruction itself. The following sequence is, for example, permissible:

```
OVER)     04   NEW   INSTR   plant new instruction in NEW
          117   0      0     dummy instruction
NEW)      04    A1     A1    gets overwritten
```

Here the dummy instruction must not be left out or the old form of the instruction in register NEW will be obeyed (unless there happens to be an interruption after the instruction labelled OVER is obeyed). A less obvious example is provided by a counting instruction (with function 80 to 83) which counts in and then jumps to another instruction but this kind of programming trick is normally to be avoided on other grounds. This restriction is necessary because the execution phases of successive instructions are overlapped, the computer extracting the next instruction from the working store before the current instruction is completed. This overwriting of the next instruction is, of course, permissible when it does not matter whether the old or the new form of the instruction is obeyed (for example, it is permissible to write into the Y-address of an immediately following 3-address 86-instruction or into a special 150-instruction, which is in any event interpreted by the Monitor Program).

2.2.23 If an address is replaced then one of the earliest processes, in obeying the instruction, is to check this address for lock-outs and reservation violation. If this address is locked out or does violate reservations then the corresponding action will take place. For example, the instruction

37 (AO-1) A100

will cause the program to be suspended because of reservation violation and not because it is an illegal instruction.

2.2.24 In some instructions the defining equation (i.e. the equation indicating the effect of the 3-address form of the instruction) does not contain a reference to all three addresses or their contents, e.g. the defining equation for the 03- instruction is z'=y and no reference is made to X or x. X, in this case, is a redundant address and when such an instruction is obeyed X is not used. The redundant address is not checked for lockouts and reservation violation unless replaced. If it is replaced because of replacement (see 2.2.23) the address will be checked for lock-outs and reservation violation but it will not be used, i.e. it is still redundant.

In the following 3-address instructions the X-address is redundant:- 03, 13, 04, 14, 23, 24, 93

In the following 3-address instructions the Y-address is redundant: 86, 87

In the following 3-address instructions the Z-address is redundant: 116, 117, 140, 141, 142

The X- and Y-addresses in some compound pairs of the 140, l42 and 142-instructions are redundant.

(i) For all devices
Mode 13 (Interrogate). The Y-address of the 142-instruction is redundant.

(ii) For all devices except the drum, and magnetic tape
Mode 16 (Disengage). The X- and Y-addresses of the 142-instruction are redundant.

(iii) For Magnetic Tape
Mode 14 (Rewind). The X- and Y-addresses of the 142-instruction are redundant.

(iv) Internal Transfer
The Y-address of the first 142-instruction is redundant.

Destination addresses of unsuccessful jumps are not checked for lockouts and reservation violation unless replaced, see 2.2.23.

The information in this section refers in general both to Orion 1 and Orion 2. Described here are the differences to be noted when reading these subsections for Orion 2.

Section 2.2.16. Because the control panel is different on Orion 2, the control number is displayed but not on Orion 2's J register.

Section 2.2.22. Orion. 2 can overwrite instructions as there is no overlap as there is on Orion 1

2.3 Modes of Operation

2.3.1 Most of this Reference Manual and the published descriptions of Orion and its programming techniques are concerned with the normal or Programmers' Mode in which the system operates. When an interruption occurs the Monitor Staticisor (also known as HKFF) is turned on and the computer is thrown into the Monitoring Mode, in which part of the Monitor Program operates; the Monitor Program returns the computer to Programmers' Mode by obeying a 152-instruction. The Monitoring Mode can be thought of as a subsidiary mode of the Programmers' Mode.

The following events occur in Monitoring Mode and not in normal Programmers' Mode.

(a) Lockouts have no effect.

(b) There are no interruptions on the completion of peripheral transfers.

(c) The timer is out of action.

(d) The reservation-checking system is inoperative, and the datum-point register is cleared (to zero).

(e) Interruptions due to peripheral incidents are delayed, by being stored in the appropriate peripheral control unit, until return to Programmers' Mode.

(f) The instructions of group 15 (see Sec. 3.15) are valid.

(g) Program failures (e.g. illegal instructions or impermissible operands) cause the instruction to be repeated - they must therefore never be allowed to occur.

(h) The 140-instruction operates differently in that its Y-address contains the machine address (k-bits) of the peripheral device concerned and not the programmer's name.

(i) The 141-instruction similarly specifies the machine-address of the drum location referred to and not the programmer's address (see Sec. 3.14).

2.3.2 There is a key-operated switch on the main control panel of the computer which can be used to put the machine into Engineering Mode. In this mode the computer behaves exactly as in the Monitoring Mode described above but with the following exceptions:

(a) Lockouts remain effective, but a locked-out instruction does not cause an interruption - it is simply repeated, over and over again (without being completed), until the lockout is lifted.

(b) The 152-instruction does not return the computer to Programmers' Mode.

(c) The 157-instruction, which causes the computer to wait (see Sec. 3.15) must not be obeyed in a part of the store which is subject to a lockout (this causes the microprogram pulse to be lost). The same is true of the 153- and 156-instructions.

(d) A set of control keys becomes operative which allow the computer to be stopped, to obey instructions set up on the handkeys and to be operated in other ways necessary for its proper maintenance (see Sec. 2.4.5).

The key to operate the switch may be inserted and withdrawn only when the switch is turned to the Normal (Programmers' Mode) position.

2.3.3 The same key-operated switch referred to in Sec. 2.3.2, above, can also be used to put the computer into Engineers' Time-Sharing Mode. The machine then behaves exactly as in normal Programmers' Mode (or Monitoring Mode, since interruptions are allowed) except that the engineers' control keys described above are operative. This mode facilitates maintenance of the time-sharing features of the system.

The information in this Section refers in general to both Orion 1 and Orion 2. Described here are the differences to be noted when reading these sections for Orion 2.

Section 2.3.1.

Subsection (d) On Orion 2 the datum point is ignored not cleared.

Section 2.3.2.

Subsection (c) On Orion 2 no pulse is lost and so these instructions may be obeyed in these circumstances.

2.4 The Main Control Desk

2.4.1 The main control desk (sometimes called the console) has a level surface, behind which is a nearly vertical panel on which are the main controls and displays for the computer. On the level surface are, on the operator's left, the main TR5 paper-tape reader (*TRA) and, the operator's right, the main Flexowriter.

2.4.2 On the vertical panel of the main control desk are a number of displays for the use of the maintenance engineers. Most of the controls are also for the engineers' use and are in fact inoperative in the normal Programmers' Mode (see Sec. 2.3.2). The only keys which are normally operative are the Handkeys and the controls for the main tape-reader and Flexowriter (see Secs. 2.4.8 and 2.4.9 below).

2.4.3 The Handkeys are a set of 48 single-acting switches arranged in two rows of 24 and spaced and labelled to correspond to the fields in a machine-instruction. The keys are also numbered 0 to 47 to show their bit-positions. We denote the keys thus: H0, H1, ..., H47. If the computer reads from pseudo-register 18 (see Sec. 2.6) a word is obtained which has 1-bits in those digital positions where the corresponding handkeys are down and 0-bits elsewhere. Pseudo-register 19 contains the inverse word (0-bits where the handkeys are down). It is strongly recommended that programs should not use the handkeys unless this is essential. These keys are primarily intended for engineers' use.

2.4.4 The displays take the form of lights, each corresponding to a bit in a register; the light is on if the bit is a one. These displays are useful only if the machine is stopped or on "slow"; they are therefore of no use in Programmers' Mode. The principal displays are the following:

(a) The Route Map. This occupies the left-hand end of the main control-panel; it is primarily of concern to engineers but includes a few lights of use to programmers, notably a red light which is on when the overflow-indicator is set, a row of 9 lights giving the datum-point and a row of lights showing the settings of the monitor indicators (to show, e.g. whether monitoring on signals is turned on, etc.)

(b) In the centre at the top of the control panel is a set of 48 lights, arranged in two rows of 24, which show the content of G or H (according to the position of a switch to the right of the display). The registers G and H are in the arithmetical unit.

(c) Below the G/H display are two rows of lights showing the contents of further registers in the arithmetic unit, viz. M and K on the left and J on the right (the upper half holding the "write address" and the lower half the control number).

(d) Below this are lights displaying the S, F, R and T bits in the current instruction. (Below these lights are the handkeys.)

(e) At the top towards the right of the control panel is a set of lights displaying the content of L, another register of the arithmetical unit.

The M and K registers are used to hold the mode and k bits, respectively, in the initiation of peripheral transfers (the k bits are the machine address of the peripheral device to be used). These registers are, however, also used for other purposes, in particular K is used to manipulate the shift-number in group 5 instructions and both M and K are used in floating-point instructions (for exponent arithmetic) and in the count instructions (80 to 83). The L register is used in multiplication and division and other instructions with double-length operands.

2.4.5 In Engineering Mode certain controls become operative which are important to those writing certain basic programs. They will therefore be described here. The chief controls are three double-acting keys. Each of these has three positions; up, central and down. The keys are the following:

(a) left-hand key: Up ("insert") - for storing the handkeys,
Central ("automatic") - for obeying stored instructions.
Down ("manual") - for obeying the handkeys.

(b) centre key: Up ("jump to 128") - sets control number to 128,
Central - normal (key is spring-loaded),
Down ("single step") - to obey one instruction.

(c) right-hand key: Up ("slow") - to obey instructions at reduced rate
Central ("stop") - to stop the computer,
Down ("run") - to obey instructions normally.

These keys are mounted centrally near the bottom of the control panel, slightly towards its right-hand side.

Certain combinations of settings of these keys are prevented from having any effect by electronic interlocks (e.g. "single-step" and "run"). In general, one should not move the left or centre keys unless the computer is stopped (i.e. with the right-hand key in its central position). In the following description we suppose that all three keys are initially in their central positions - this will be called the Basic Position.

The control number is displayed in the lower half of J (see 2.4.4c above). Starting from the Basic Position, if the right-hand key is depressed (to "run") the computer will obey instructions from the working store, starting at the address displayed. Returning this key to its central position ("stop") at any time will cause the machine to stop as soon as it has completed the instruction (or compound instruction - see the Glossary, Sec. 0.3) it was obeying at that instant.

Pushing the centre key up ("jump to 128") while the computer is stopped will set both halves of J to 128, i.e. it will set both the control number and the write address to 128. This key springs back to its central position when released. If this key is pressed down ("single step") from the Basic Position, then the computer will obey a single instruction (or compound instruction) from the working store at the address equal to the control number; the control number will also be altered to the address of the next instruction (normally by having one added to it but it will be increased by more if a compound instruction is obeyed and will be set to a new value by a successful jump). In the case of most of the instructions the result written into the store will at this point be held in register H and can be displayed on the top-most set of lights (see Sec. 2.4.4b above). The write address, displayed in the upper half of J, shows where this result was written. The S, F, R and T bits of the next instruction are also displayed at this moment (see Sec. 2.4.4d) and the whole instruction is in register G, which can be displayed instead of H if desired.

Pushing the right-hand key up (to "slow") has the same effect as a series of single step operations following one another at a rate which can be altered by an adjacent knob. A program can be obeyed slowly by using this feature (but 157-instructions will not then cause a stop). It is very important not to try to go too fast when on "slow" as a long instruction (e.g. 142 pair) might then go wrong.

If, starting from the Basic Position, the left-hand key is pushed down (to "manual") then the computer is ready to obey machine instructions set up on the handkeys - these are called manual instructions. Such instructions are normally obeyed one at a time by using single step operations but they can, exceptionally, be obeyed on "slow" or "run". The control number will not be changed by obeying manual instructions (other than successful jumps) so that it is possible to stop a program, obey a manual instruction and then continue with the program. It is not possible to obey compound instructions manually.

Manual instructions may be used to display the content of any working store register. For example, to display the word in the register with machine-address 33 we stop the computer, set up the machine instruction

```
04X    0    33
```

go to "manual" and do a single step operation. The word required is then in H and can be displayed on the upper set of lights.

If one starts from the Basic Position and pushes up the left hand-key (to "insert") then the computer is ready to insert (i.e. store) the word set up on the handkeys into the working store register whose machine address is in the upper half of J (the write address): it will actually insert the word when a "single step" is given; the write address is increased by one immediately afterwards so that another instruction can be set up and inserted into the next register. In this way one can easily insert short sequences of instructions into the working store; this is the foundation of the "bootstrap procedure" (see Sec. 2.4.6 below). It will be recalled that pushing up the centre key ("jump to 128") sets the write address to 128, as well as the control number. (The "insert" process also operates on "slow", which is always equivalent to a succession of single steps.)

To insert words from the handkeys into an address other than 128 it is necessary to be able to set the write address to an arbitrary value. This can be done by obeying a 2-address 74-instruction, which has the effect of setting the write-address equal to the address set up in X on the handkeys. If this is preceded by a manual 2-address 75-instruction to the same address then both halves of J will be set to this address. (Note that the 75 must precede the 74 since a successful jump interchanges the two halves of J.)

2.4.6 The 'bootstrap' procedure is used in Engineering Mode to get programs into the computer without using the built-in programs. This is necessary during commissioning and maintenance. The procedure is usually preceded by a manual 143-instruction to clear the working store.

A program to be read in by the bootstrap procedure must be punched in binary as machine instructions; the paper tape (7-track) or cards will therefore have been prepared earlier by computer. The following description assumes that the program is punched in 7-track paper tape. The basis of the procedure is that a single compound instruction of the form

```
140.2    0    K
142      W    N
```

can read the whole program into the working store - it will in fact read in N characters from peripheral device K and place them in the store starting at machine address W. The peripheral device is normally the main tape reader but the values to be used for W and N will depend on the program being read in. In order to avoid having to use a different procedure for each program a standard procedure is used which reads in a single 142-instruction (containing the variable data) which is punched at the beginning of the program.

This standard procedure is to insert into 128 and 129 the instructions

```
140.2     0     K        (In 128)
142      129    8        (In 129)
```

which are then obeyed. These read 8 characters (i.e. a word) from peripheral device K (the main tape-reader) into 129, i.e. over the 142-instruction above. These 8 characters represent the new 142 appropriate to the program being read in. We now have in 128 and 129 the compound instruction needed to read in the program, and this is obeyed.

The standard bootstrap procedure is therefore as follows:

(1) Go to Basic Position (all three keys central).

(2) Load tape in main tape reader with the first non-UC character in the reading position.

(3) Left and centre keys both up (i.e. "insert" and "jump to 128"); centre key springs back.

(4) Set up on the handkeys the instructions
140.2 0 K

(5) Centre key down ("single step").

(6) Set up on the handkeys the instruction:
142 129 8

(7) Centre key down ("single step").

(8) Left key central ("automatic").

(9) Centre key up, then down ("jump to 128" and "single step").

(10) Repeat step 9.

Notes on this procedure (H12 means handkey no. 12):

Step 3: the write address is set to 128.
Step 4: the mode is in the m.s. five bits of X so H12 should be down.
Step 5: the key will spring back when it is released; this operation causes the 140-instruction to be inserted in 128.
Step 7: the 142 is inserted into 129.
Step 8: prepare to obey stored instructions.
Step 9: set jump address to 128 and obey the compound instruction to read new 142 from tape into register 129. Tape in reader moves.
Step 10: obey new form of compound instruction in 128 and 129 to read in the program.

Many of the programs which are read in by means of this procedure are arranged to start in 128 and can therefore be entered by "jump to 128" and "run" on the centre and right-hand keys, respectively. A program which has to be entered at some other machine address can be entered by obeying a manual 75-instruction.

2.4.7 If the computer encounters a 157-instruction in Engineering Mode it will stop, displaying its effective addresses. The computer can be caused to continue at the next instruction by moving the right-hand key to its central position ("stop") followed by either "run" (right-hand key down) or "single step" (centre key down).

2.4.8 Near the bottom of the main control panel, slightly to the left of centre is a group of controls and displays for the main tape-reader and Flexowriter. There is a row of small rectangular lights for each device; some of these lights serve also as push-buttons. As a general rule a button may be pressed if it is lit up in white. The upper row are the controls for the main Flexowriter and consists of four lights as follows (from left to right):-

(a) "Input Selected". This light is normally white; it turns blue when the computer is reading from the Flexowriter keyboard; the white light on the Flexowriter is then also lit. If pressed it calls in the Monitor Program to read from the Flexowriter.

(b) "Output Selected". This white light turns blue when the computer is typing (and possibly punching) on the Flexowriter.

If this light is pressed it causes run out (repeated punching of UC) provided the Flexowriter is disengaged.

(c) "Engage". When white this may be pressed to engage the Flexowriter; it then turns green.

(d) "Disengage". This is red if the Flexowriter is disengaged. It is otherwise white, when pressing it will disengage the Flexowriter (e.g. to allow the stationery to be changed).

A "Select" control is also mounted on the Flexowriter itself. This is marked "Accept".

2.4.9 The lower row of lights are the displays and controls for the main tape-reader. These consist of four lights as follows (from left to right):-

(a) 7-track/5-track. This control is divided into two halves. The upper half displays "7-track" in yellow when the "associated electronics" of the tape-reader are set for 7-track tape. When set for 5- track tape the lower half of the light displays "5-track" in blue. Pressing this light switches the associated electronics from one state to the other (the tape reader itself has to be mechanically adjusted as well).

(b) "Select". This white light is pressed to call in the Monitor Program to read the heading of a tape (see Sec. 5.7). This light turns blue when the reader is selected, i.e. is actually being used by the computer.

(c) "Engage". This button is green when the tape-reader is engaged. When white it may be pressed to engage the reader.

(d) "Disengage". This is red when the tape-reader is disengaged. When white it may be pressed to disengage the reader (for example to change tapes).

The "Disengage" button may be used if a knot of tape is approaching the reader; the "Engage" button can then be pressed to allow reading to continue.

The detail in this section refers only to Orion 1. The control panel for Orion 2 and the actual operation of the machine from the panel (in Engineers Mode) is different. The naming of the control registers for Orion 2 is not the same as Orion 1, so that Orion 2 J's register is not the same as Orion 1's.

The sections that do apply to Orion 2 are 2.4.1., 2.4.2 and, 2.4.8. except for the position of the lights.

Pseudo-Registers

2.6.0 The pseudo-registers are a number of auxiliary registers containing useful constants or the state of switches inside or outside the machine. They can be read from but it is not possible to write to them. They are referred to by the instructions 20-27, 66, 67 and 112 (see sections 3.2, 3.6 and 3.11) - these instructions take the same time as the corresponding instructions (00-07, 60, 61 and 110) for ordinary registers. Pseudo-registers are available to all programs impartially - they are not subject to lockout or reservation checking. The Y^{th} pseudo-register is referred to as PY, and its contents as pY.

2.6.1 Contents of Pseudo-registers

The contents of even-numbered pseudo-registers from.P0 to P18 are given below; for odd number registers p(2n+1) = ¬p(2n), i.e. the contents of each odd numbered pseudo-register is the inverse of the contents of the preceding even numbered register.

p0 = 0

p2 = Local civil time - see section 2.6.3 for details of the code.

p4 = Zero if overflow clear, all ones if overflow set. Any instruction referring to p4 or p5 clears overflow.

p6 is as p4 except it is not cleared when used.

p8 = upper half word mask - i.e. 24 ones followed by 24 zeros.

p10 = -1.0 i.e. one followed by 47 zeros.

p12 = ½, i.e. a zero, a one followed by 46 zeros.

p14 = Mask for X address, i.e. 9 zeros, 15 ones, 24 zeros.

p16 = Mask for Z address, i.e. 26 zeros, 6 ones, 16 zeros.

p18 = Handswitches. The handswitches are a set of 48 keys which are provided for the use of the engineers; their use by programmers is not recommended except in very special circumstances - e.g. Pegasus simulator. It is the operator's responsibility to ensure that only one program using the handswitches is in the machine at once.

2.6.2 Unallocated Pseudo-Registers

At present pseudo-registers 20-31 inclusive contain zero for even numbers and all ones for odd numbers. Pseudo-register numbers above 31 are taken modulo 32, i.e. only the least significant five bits are decoded. Programmers are strongly recommended not to use either of these facts as they may be changed if it is decided to add further pseudo-registers to the machine.

2.6.3 Local Civil Time

The digital clock in Orion is a 24-hour clock in hours, minutes and seconds. It is stored in a one out of n code, i.e. in each field which has n possible values one and only one of n bits is a 1. In each case the m.s. bit of the field represents 0, the next 1, the next 2 and so on. The fields are as follows:

D0	to	D2	tens of hours
D3	to	D12	hours
D13	to	D18	tens of minutes
D19	to	D28	minutes
D29	to	D34	tens of seconds
D35	to	D44	seconds
D45	to	D47	not used (always zero)

e.g. the time 17.08.23 is represented by 1-bits in digits:

1, 10, 13, 27, 31, 38

and 0-bits elsewhere.

Third Character
(Twelves of Hours)

Value	Binary Code	Decimal Code
0	000000	0
1	000001	1

Fourth Character
(Hours)

Value	Binary Code	Decimal Code
0	000000	0
1	000001	1
2	000011	3
3	000010	2
4	000110	6
5	000111	7
6	001111	15
7	001110	14
8	001010	10
9	001011	11
10	001001	9
11	001000	8
0	001000	8
1	001001	9
2	001011	11
3	001010	10
4	001110	14
5	001111	15
6	000111	7
7	000110	6
8	000010	2
9	000011	3
10	000001	1
11	000000	0

Fifth Character
(Tens of Minutes)
Seventh Character
(Tens of Seconds)

Value	Binary Code	Decimal Code
0	000000	0
1	000001	1
2	000011	3
3	000111	7
4	000101	5
5	000100	4
0	000100	4
1	000101	5
2	000111	7
3	000011	3
4	000001	1
5	000000	0

Sixth Character
(Minutes)
Eighth Character
(Seconds)

Value	Binary Code	Decimal Code
0	000000	0
1	000001	1
2	000011	3
3	000111	7
4	000101	5
5	001101	13
6	001111	15
7	001011	11
8	001001	9
9	001000	8
0	001000	8
1	001001	9
2	001011	11
3	001111	15
4	001101	13
5	000101	5
6	000111	7
7	000011	3
8	000001	1
9	000000	0