ACC NEWSLETTER VOL 3 ISS 4 OCT 75

HITS

The US magazine Popular Electronics has published (Sep 75) details of a proposed standard for recording digital computer data on a cheap cassette recorded.

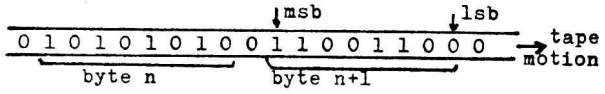
Known as the Hobbyists Interchange System, it is designed to allow amateur computer enthusiasts to exchange programs. Simplicity, reliability and low cost are emphasised rather than speed (approx 30 bytes/sec).

We suggest that the system is adopted by the ACC.

Format

All data is written in 8 bit ASCII code bytes or, by agreement between sender and recipient, in 8 bit object code.

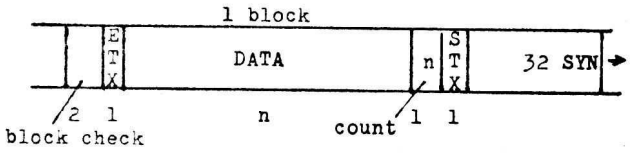
Each 8 bit byte is written in bit serial form, least significant bit first, followed by a single binary 0 (for timing purposes).When using a stereo recorder data is recorded onto the right hand channel and the left hand channel is blank.



Data is recorded in 'blocks' of 1 - 255 data bytes. Each block consists of;

- at least 32 ASCII SYN characters (synchronising code 00010110), followed by;
- a single ASCII STX (Start of Text 00000010)
- an 8 bit count word containing (in binary) the number of true data bytes (i.e. excluding SYN, STX, Count word, ETX & Block Check) in the block.
- the actual message then follows and is terminated by;
- a single ASCII ETX (End of Text 00000011) followed by;
- two Block Check characters. These are normally zero but can be used to hold an error detection count (CRC, check sum etc.). If they are used the writer of the tape must provide a program for the machine of interest which will read and utilise them. This 'bootstrap' program should appear at the front of the tape and should be terminated by an 'end of file' block.

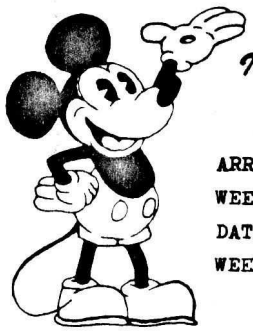
An End of File block is one with zero data bytes (Count = 0)



Note that every 8 bit byte (including SYN, STX etc) is followed by a binary 0 when recorded.

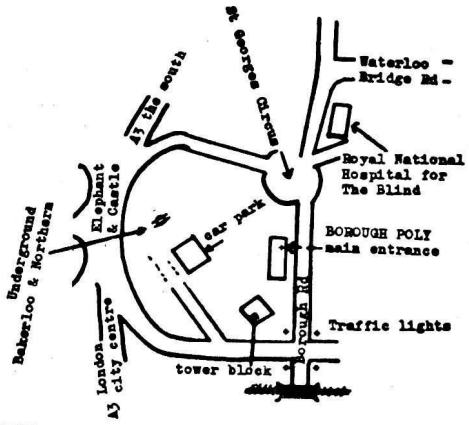
Recording

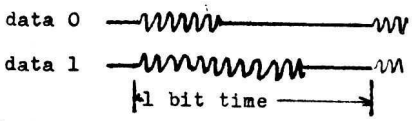
Each bit occupies a defined time period (2.75 mS is the standard but faster recording rates are possible on high quality equipment for the user's own operations). A burst of 2kHz tone is recorded starting at the beginning of the bit time and extending for approx 1/3 of the bit time for a 0, 2/3 for a 1.



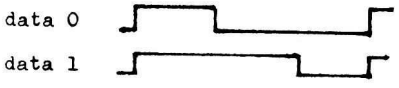
IN THIS ISSUE

WEENY-BITTER SEMINAR 29 Nov 7.0 pm
Borough Rd Polytechnic. London SE1
COMMITTEE MEETING after the seminar





To read data from the tape the tone bursts are rectified to give square waves of different mark to space ratio for 1's and 0's. The mark-space ratio is then determined by counting at a regular rate, starting at the beginning of the pulse and counting up until the end of the pulse, then counting down at the same rate until the start of the next pulse. If the final count is positive the bit is a 1, if negative it is a 0. This technique is extremely tolerant of tape speed and recording parameters.



As usual one can make a trade off between soft and hardware techniques for doing the recording and retrieval. A minimum hardware approach uses software generated 2kHz square wave bursts which are connected to the cassette recorder via a simple low pass filter, and an amplifier /rectifier to pick the data from the recorder output and present it to a single bit input port to the computer. Data formatting and unformatting, generation of the 2kHz bursts, decoding the rectified bursts into 1's and 0's, receive synchronisation and simple error detection is all done with software - about 300 bytes on an 8080.

ARRAYS

P.J.Rodman

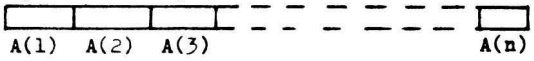
For simple programs, the programmer only needs simple variables to contain his program data. For more complex programs it is convenient to have a large, fixed amount of data storage which can be accessed under a single variable name, usually with an index of some sort (c.f. index sets in mathematics). If the storage is read from it only, it is usually called a TABLE. If the program can write to it as well, then it is called an ARRAY. Sometimes it is convenient to have an array which is 2 or 3 dimensional, e.g. for a computer simulated game of chess, a 2 dimensional 8 by 8 array could be used as a board. Arrays are fairly common in high level computer languages such as FORTRAN and ALGOL, but at the machine level an array must be programmed using only primitive operations. Implementing arrays at machine level will be discussed after a short theoretical discussion.

A formal definition of an array (recursive) is;

A 1-dimensional array is a vector whose elements are scalars (numbers).

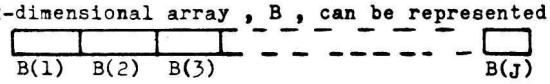
For n greater than one, an n-dimensional array is a vector whose elements are (n-1)-dimensional arrays with identical index sets.

This odd sounding definition is actually quite straight forward. A 1-dimensional array , A , can be represented as;

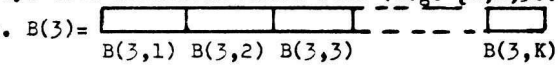


where A(1), A(2), . . A(n) are simple numbers (or storage locations). The set {1,2,3 . . n} is called the index set of array A. Array A is said to have n elements.

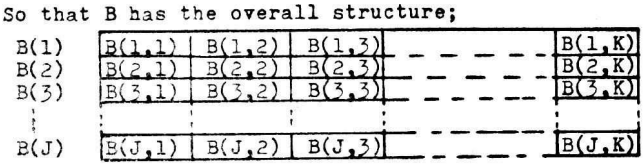
A 2-dimensional array , B , can be represented as;



where B(1),B(2) . . . B(J) are one dimensional arrays with identical index sets (e.g. {1,2,3.. K})



So that B has the overall structure;



The set $\{(m,n) \mid m=1,2,\ldots,J \; ; \; n=1,2,\ldots,K\}$ is the index set of array B, and array B is said to be a "J x K array".
This definition can be extended to n-dimensional arrays.

These structures are obviously useful for storing tables, matrices etc. On virtually all machines, the main mamory is a linear memory, i.e. each location has two, and only two, neighbouring memory locations (apart from the first and last locations which have only one neighbour). Thus it is not possible to directly implement n-dimensional arrays for n greater than one. They are usually stored in consecutive memory locations to simplify the mapping function from the index set to the array as stored in the memory.

For one-dimensional arrays the mapping is a simple one. Let the index set be 1,2,...,n (e.g. as in FORTRAN). If we wish the array to be stored in locations 1 to n of the memory, then the j-th element of the array would be stored at location j. More generally, if the array is to be stored in locations k to k+n-1 (i.e. n locations), then the j-th element would be stored at location (j-1)+k.

(This can easily be verified by "plugging in" some values for j,n and k).

This can be extended if each element consists of more than 1 word of memory. e.g. for 3 words per element words belonging to the j-th element would be at locations

3j+k-3 , 3j+k-2 and 3j+k-1

Generally, for m words per element, the words belonging to the j-th element would be in locations;

mj+k-m , mj+k-m+1 , ... mj+k-1

Finally, the general equation for the j-th element of an array with index set {p,p+1,...,q} , m words per element, and starting in memory location k is;

m(j-p)+k , m(j-p)+k+1, . . ,m(j-p)+k+m-1

EXAMPLE:

Write a routine to handle a 50 element, one dimensional array, 1 word/element, starting in location 320. The elements will be numbered 1,2,..,50.

Solution:

In the above notation, n=50, k=320, m=1, p=1, q=50
The array resides in locations 320 to 369
The j-th element is

1.(j-1)+320 = j+319

We then have

```
1.           LOAD    J         J IS ELEMENT REQUIRED
2.           ADD     NUM       ADD 319
3.           STORE   ADDRESS
4.           LOAD*   ADDRESS   LOAD ELEMENT J INTO ACC
5. NUM       DATA    319
6. ADDRESS   DATA    0
```

(Note; the * in line 4 denotes indirect addressing. If the machine being used has no indirect addressing capabilities, then ADDRESS could be loaded to an index register and addressing done via that register.)

For a 2-dimensional M x N array, the structure is similar, except that instead of single words for elements, we have an M element one-dimensional array with N words/element. If the index set was

(j,k) j=1,2,..,M ; k=1,2,...,N

the location of the (i,j)th element would be

(i-1)N + (j-1) + k.

This can be extended easily to m words/element.

Similarly for a L x M x N array (3-dimensional) the location of the (p,q,r)th element is

(p-1)MN + (q-1)M + (r-1) + k

and so on for higher dimension arrays.

It should be noted that for n greater than one dimensioned arrays, and one-dimension arrays with more than one word/element, multiplication is necessary. If the array sizes are an integral power of 2 however, say 2^R, then shifting a register k places towards its most significant bit effectively multiplies it by 2^R, and so the multiplication operation becomes unneccessary.

Other methods can be used, such as the Dope, or Iliffe, vector methods, which will not be discussed here (1), (2).

References and Bibliography

(1) Compiling Techniques, F.R.A.Hopgood. Macdonald/American Elsevier Computing Monographs.

(2) Information Representation and Manipulation on a Computer, E.S.Page and L.B.Wilson. Cambridge University Press

(3) The Art of Computer Programming. Vol 1 D.Knuth Addison-Wesley

WEENY-BITTER SIMULATOR

```
"BEGIN" "COMMENT"SIMULATOR FOR WB-2;
        "INTE""ARRAY"CORE[0:1023];
        "BOOL"C,Z,P;
        "INTE"INPORT,OUTPORT;

        "INTE""PROC"MODE(B76,N);"VALUE"B76,N;"INTE"B76,N;
                "BEGIN" "INTE"P,Q;
                        P:=CORE[N];
                        "IF"B76=0"THEN"MODE:=P
                        "ELSE""IF"B76=1"THEN"MODE:=
                                        256*CORE[P+1]+CORE[P]
                        "ELSE"  "BEGIN" Q:=256*CORE[3]+CORE[2]+P-128;
                                        MODE:="IF"B76=2"THEN"Q
                                        "ELSE"256*CORE[Q+1]+CORE[Q]
                                "END"
                "END"MODE;

        "PROC"PR(Q);"VALUE"Q;"INTE"Q;
                "CODE"  %LD     $Q
                        %LDR:L  $0
                        %JIL    $162;

        "INTE""PROC"IODER(P,Q);"VALUE"P,Q;"INTE"P,Q;
                "BEGIN" "INTE"R;
                        "CODE"  %BL     $P
                                %MORR
                                %ST     $R;
                        IODER:=R
                "END"IODER;

        "INTE""PROC"UND(P,Q);"VALUE"P,Q;"INTE"P,Q;
                "BEGIN" "INTE"R;
                        "CODE"  %LD     $P
                                %AND    $Q
                                %ST     $R;
                        UND:=P
                "END"UND;

        "INTE""PROC"XODER(P,Q);"VALUE"P,Q;"INTE"P,Q;
                XODER:=IODER(P,Q)-UND(P,Q);

        "PROC"TEST(X);"INTE"X;
                "BEGIN" X:=UND(X,511);
                        "IF"X>255"THEN"
                                "BEGIN" X:=X-256;
                                        C:="TRUE"
                                "END"
                        "ELSE"C:="FALSE";
                        P:=X<128;
                        Z:=X=0
                "END"TEST;

        "INTE""PROC"NEXT;
                "BEGIN" "INTE"PC;
                        PC:=NEXT:=256*CORE[3]+CORE[2];
                        PC:=PC+1;
                        CORE[3]:=PC DIV"256;
                        CORE[2]:=UND(PC,255)
                "END"NEXT;

        "PROC"PUSH(N);"VALUE"N;"INTE"N;
                "BEGIN" "INTE"SP;
                        SP:=256*CORE[5]+CORE[4]+1;
                        CORE[SP]:=N;
                        CORE[5]:=SP"DIV"256;
                        CORE[4]:=UND(SP,255)
                "END"PUSH;

        "PROC"POP(X);"INTE"X;
                "BEGIN" "INTE"SP;
                        SP:=256*CORE[5]+CORE[4];
                        X:=CORE[SP];
                        SP:=SP-1;
                        CORE[5]:=SP"DIV"256;
                        CORE[4]:=UND(SP,255)
                "END"POP;

        "INTE""PROC"ASCII(INT);"VALUE"INT;"INTE"INT;
                ASCII:= "IF"INT=2"THEN"10
                        "ELSE""IF"INT=32"THEN"96
                        "ELSE""IF"INT=77"THEN"13
                        "ELSE"INT+32;

        "INTE""PROC"INT(ASCII);"VALUE"ASCII;"INTE"ASCII;
                INT:=   "IF"ASCII=10"THEN"2
                        "ELSE""IF"ASCII=96"THEN"32
                        "ELSE""IF"ASCII=13"THEN"77
                        "ELSE"ASCII-32;
```

This program, written in Algol 60, simulates the operation of the WB-2. The core of the machine is represented by an integer array CORE(0:1023), which is loaded with program before the simulation begins by the procedure LOAD. A dump of CORE is provided just before the start of simulation, when a NOP instruction is executed, and when the simulator halts. The addresses of the input and output ports are 007 and 010 respectively. The condition codes C,P and Z are represented by the corresponding boolean variables C,P,Z. In ICL 4130 ALGOL, key words may be represented by their first four characters; e.g. procedure is "PROC" integer is "INTE". The quotes araound key words are required by the compiler, which ignores blanks and newlines everywhere outside strings.

MODE (B76,N) interprets N as the address of a word of CORE which specifies, directly or indirectly, the address of the operand and unravels it according to the top two bits (B76) of the current instruction to yield the actual address of the operand.

PR(Q) is a code procedure which prints the character represented internally by Q on the line printer.

IODER(P,Q) UND(P,Q) XODER(P,Q) perform the logical IOR,AND and XOR respectively of their arguments. The pseudo-German names are used because of a peculiarity of out compiler which would object to the use of the labels IOR, AND, XOR somewhere else in the program if these were also procedure names.

TEST(X) sets C,P and Z according to the value of X. Some operations (e.g. SUB) can produce a result which overflows into the bits above bit 8 (e.g. 0 - 1 = 111 . . 11) so TEST clears these high bits. Bit 8 is used to indicate a carry, so this bit is reset and C is set to true if bit 8 is found to be set.

NEXT yields the address of the word pointed out by the PC, i.e. the value of the PC, and then increments the PC.

PUSH(N) POP(X) manipulate the stack.

ASCII(INT) yields the ASCII character code for the internal character INT

INT(ASCII) yields the internal character code for the ASCII character ASCII.

```
"INTE""PROC"INOCT;
        "BEGIN" "INTE"N,Q;
                N:=0;
L1:             ADVANCE(6);
                Q:=DECODE(6);
                "IF"Q*(Q-2)=0"THEN""GOTO"L1;
L2:             "IF"Q=10"THEN"INOCT:=-1
                "ELSE""IF"(Q-16)*(Q-23)>0"THEN"
                        "PRINT"''L`ILLEGAL NUMBER READ `,
                        'WHILE LOADING'L``,STOP
                "ELSE"  "BEGIN" N:=8*N+(Q-16);
                                ADVANCE(6);
                                Q:=DECODE(6);
                                "IF"Q*(Q-2)"NE"0"THEN"
                                        "GOTO"L2;
                                INOCT:=N
                        "END"
        "END"INOCT;
```

INOCT reads an octal number from the card reader. Numbers are delimited by spaces or newlines ; excess delimiters are ignored. If a star is read, INOCT takes the value -1.

```
"PROC"MEMDUMP;
        "BEGIN" "INTE"J;
                "PRINT"''F`CORE DUMP'L2S10``;
                "FOR"J:=0"STEP"1"UNTIL"15"DO"
                        "PRINT"OUTOCT(2,J),''S2``;
                "PRINT"''L``;
                "FOR"J:=0"STEP"1"UNTIL"1023"DO"
                        "BEGIN" "IF"UND(J,15)=0"THEN"
                                    "PRINT"''L``,OUTOCT(4,J),
                                    ''S6``;
                                "PRINT"OUTOCT(3,CORE[J]),''S``
                        "END";
                "PRINT"''L2``
        "END"MEMDUMP;
```

MEMDUMP prints a dump of the contents of CORE.

```
"PROC"LOAD;
        "BEGIN" "INTE"S,N;
LOOP:           S:=INOCT;
                "IF"S=-1"THEN""GOTO"R;
                "IF"S*(S-1023)>0"THEN""PRINT"''L`LOADING `,
                        'INTO ILLEGAL CORE'L``,STOP;
                N:=INOCT;
                "IF"N*(N-255)>0"THEN""PRINT"''L`ILLEGAL `,
                        'NUMBER WHILE LOADING'L``,STOP;
                CORE[S]:=N;
                "GOTO"LOOP;
R:      "END"LOAD;
```

LOAD reads pairs of numbers using INOCT , and assigns the value of the second to the word of CORE specified by the first. A star terminates loading.

```
"PROC"MISC(B210);"VALUE"B210;"INTE"B210;
        "BEGIN" "SWITCH"OP:=HLT,NOP,CLC,SEC,ION,IOF,RTS,RTI;
                "GOTO"OP[B210+1];
HLT:            "PRINT"''L`HALTED`,MEMDUMP,STOP;
NOP:            "PRINT"''L`NOPDUMP`,MEMDUMP;
                "GOTO"R;
CLC:            C:="FALSE";             "GOTO"R;
SEC:            C:="TRUE";              "GOTO"R;
ION:            "PRINT"''L`ION'L``;     "GOTO"R;
IOF:            "PRINT"''L`IOF'L``;     "GOTO"R;
RTS:
RTI:            POP(CORE[3]);
                POP(CORE[2]);
R:      "END"MISC;
```

MISC(B210) executes one of the eight miscellaneous instructions. STOP causes a return to the operating system.

```
"PROC"JUMP(B210,N,SUB);"VALUE"B210,N,SUB;"INTE"B210,N;
                        "BOOL"SUB;
        "BEGIN" "SWITCH"OP:=GTO,GTO,GIZ,GNZ,GPL,GMI,GCS,GCC;
                "GOTO"OP[B210+1];
GTO:            "IF"SUB"THEN"
                        "BEGIN" PUSH(CORE[2]);
                                PUSH(CORE[3])
                        "END";
                CORE[3]:=N"DIV"256;
                CORE[2]:=UND(N,255);
                "GOTO"R;
GIZ:            "GOTO""IF"Z"THEN"GTO"ELSE"R;
GNZ:            "GOTO""IF""NOT"Z"THEN"GTO"ELSE"R;
GPL:            "GOTO""IF"P"THEN"GTO"ELSE"R;
GMI:            "GOTO""IF""NOT"P"THEN"GTO"ELSE"R;
GCS:            "GOTO""IF"C"THEN"GTO"ELSE"R;
GCC:            "GOTO""IF""NOT"C"THEN"GTO"ELSE"R;
R:      "END"JUMP;
```

JUMP(B210,N,SUB) executes one of the seven jump instructions. If the boolean SUB is true, a link is first pushed on the stack if the conditions are satisfied. N is the destination address.

```
"PROC"ISIN(N);"VALUE"N;"INTE"N;
        "IF"N=INPORT"THEN"
                "BEGIN" ADVANCE(6);
                        CORE[N]:=ASCII(DECODE(6))
                "END";
```

ISIN(N) does nothing unless N is the address of the input port, in which case it reads a character from the card reader into CORE(N) . ADVANCE & DECODE are system-defined procedures.

```
        "PROC"ISOUT(N);"VALUE"N;"INTE"N;
                "IF"N=OUTPORT"THEN"PR(INT(CORE[N]));

        "PROC"OUTOCT(D,N);"VALUE"D,N;"INTE"D,N;
                "IF"D>0"THEN"
                        "BEGIN" OUTOCT(D-1,N"DIV"8);
                                PR(16+UND(N,7))
                        "END";

        "PROC"SINGLE(B210,N);"VALUE"B210,N;"INTE"B210,N;
                "BEGIN" "INTE"Q;
                        "SWITCH"OP:=CLA,INC,DEC,ADC,TST,INV,SHR,SHL;
                        ISIN(N);
                        "GOTO"OP[B210+1];
        CLA:            CORE[N]:=0;                 "GOTO"R;
        INC:            CORE[N]:=CORE[N]+1;         "GOTO"R;
        DEC:            CORE[N]:=CORE[N]-1;         "GOTO"R;
        ADC:            "IF"C"THEN"CORE[N]:=CORE[N]+1;
        TST:            "GOTO"R;
        INV:            CORE[N]:=255-CORE[N];       "GOTO"R;
        SHR:            Q:=CORE[N];
                        "IF"UND(Q,1)=1"THEN"Q:=Q+512;
                        CORE[N]:=Q"DIV"2;
                        "GOTO"R;
        SHL:            CORE[N]:=CORE[N]*2;
        R:              TEST(CORE[N]);
                        ISOUT(N)
                "END"SINGLE;

        "PROC"DOUBLE(B210,N1,N2);"VALUE"B210,N1,N2;"INTE"B210,N1,N2;
                "BEGIN" "SWITCH"OP:=MOV,ADD,SUB,AND,IOR,XOR,BIT,CMP;
                        "INTE"T;
                        ISIN(N1);
                        "GOTO"OP[B210+1];
        MOV:            CORE[N2]:=CORE[N1];             "GOTO"R;
        ADD:            CORE[N2]:=CORE[N2]+CORE[N1];    "GOTO"R;
        SUB:            CORE[N2]:=CORE[N2]-CORE[N1];    "GOTO"R;
        AND:            CORE[N2]:=UND(CORE[N2],CORE[N1]);
                        "GOTO"R;
        IOR:            CORE[N2]:=IODER(CORE[N2],CORE[N1]);
                        "GOTO"R;
        XOR:            CORE[N2]:=XODER(CORE[N2],CORE[N1]);
                        "GOTO"R;
        BIT:            T:=XODER(CORE[N2],CORE[N1]);
                        "GOTO"L;
        CMP:            T:=CORE[N2]-CORE[N1];
        L:              TEST(T);            "GOTO"RR;
        R:              TEST(CORE[N2]);
        RR:             ISOUT(N2)
                "END"DOUBLE;

        "PROC"OBEY(N);"VALUE"N;"INTE"N;
                "BEGIN" "INTE"B76,B543,B210;
                        "SWITCH"GROUP:=L0,L1,L2,L3,L4,L5,L6,L7;
                        B76:=N"DIV"64;
                        B543:=UND(N"DIV"8,7);
                        B210:=UND(N,7);
                        "GOTO"GROUP[B543+1];
        L0:             "IF"UND(B76,1)=1"THEN"SINGLE(B210,0)
                                "ELSE"MISC(B210);
                        "GOTO"R;
        L1:             DOUBLE(B210,NEXT,0);                "GOTO"R;
        L2:             DOUBLE(B210,MODE(B76,NEXT),0);      "GOTO"R;
        L3:             JUMP(B210,MODE(B76,NEXT),"FALSE");
                        "GOTO"R;
        L4:             SINGLE(B210,MODE(B76,NEXT));        "GOTO"R;
        L5:             DOUBLE(B210,NEXT,MODE(B76,NEXT));
                        "GOTO"R;
        L6:             DOUBLE(B210,0,MODE(B76,NEXT));      "GOTO"R;
        L7:             JUMP(B210,MODE(B76,NEXT),"TRUE");
        R:      "END"OBEY;

        "BEGIN" "INTE"J;
                "FOR"J:=0"STEP"1"UNTIL"1023"DO"CORE[J]:=0
        "END";
        CORE[2]:=32;
        CORE[4]:=191;
        CORE[5]:=3;
        C:=Z:=P:="FALSE";
        INPORT:=7; OUTPORT:=8;
        "PRINT"'('L'INITIAL STATE OF CORE IS',LOAD,MEMDUMP;
RUN:    OBEY(CORE[NEXT]);
        "GOTO"RUN
"END"PROG;
```

ISOUT(N) does nothing unless N is the address of the output port, in which case it prints the character in CORE(N) on the line printer.

OUTOCT(D,N) is a recursive procedure which outputs the D least significant digits of N in octal.

SINGLE(B210,N) executes one of the eight single-operand instructions, with N as the address of the operand.

DOUBLE(N1,N2) executes one of the eight double-operand instructions, with N1 and N2 the addresses of the first and second operands respectively.

OBEY(N) obeys the instruction which is N.

The MAIN PROGRAM initialises CORE by setting the PC to 40, the SP to 1677 and the rest of CORE to zero, before clearing C,Z & P and sitting in a loop fetching and obeying instructions using NEXT & OBEY.

I've tested the program and it seems to be working correctly, but there are probably a few bugs left in it so I'd appreciate some feedback from readers who manage to get it to work. If you would like a paper tape of the program, please write to me at the address below, enclosing an appropriate amount for postage and I'll send you one, by return of post if possible.

ANTHONY FISHER
DEPT COMPUTER SCIENCE
UNIVERSITY COLLEGE OF WALES
ABERYSTWYTH , DYFED

DATABASES

P.Grave

The main use of a database is in a Management Information System. Database technology is combined with a communications network (IBM refer to this as DB/DC) so that data can be fed into the system from terminals situated as close as possible to the data source -- in offices, stockroom and on the shop floor. The MIS uses the data to create a model of the company. Like a model aircraft in a wind tunnel the computer model is easier to observe than its real life equivalent and like a game of Monopoly experiments can be carried out without risking real money (well, only the cost of the computer).

ICL used their PEARL database system to help in the writing of computer programs. A large operating system such as System B is full of complicated interactions between subroutines; by storing details of these in a database it is easier to see the effects of any possible design changes.

Definitions of a database vary; let us say a database is characterised by;

1. large amounts of data
2. direct access (i.e. disc)
3. data is shared by different users & programs

A database represents the entities that are of interest to the database owner by means of records. A record is subdivided into fields each of which contains a value describing some specified attribute of the entity associated with the record. An insurance company might have records with fields called NAME, ADDRESS, POLICY-NUMBER, SUM-INSURED, BIRTH)YEAR; a particular record could contain these values:

CALDER, J
1 HIGH ST, PUTNEY
8983506
8000
1938

The search techniques employed have to cope with such requests as:

DISPLAY SUM-INSURED > 99999 & BIRTH-YEAR < 1915

One way of retrieving records satisfying some given criteria would be to search serially through the entire database. This is fine if a large batch of queries are processed in one pass, but usually takes too long.

To reduce the size of the search an index can be employed. The records are held in order, using a selected field e.g. in alphabetical order of NAME. Every n th record has an entry in the index giving its disc address. To find the CALDER, J record, the nearest preceeding value is found in the index (by a smaller serial search) ; the remainder of the index entry gives the address to start searching the main records. If necessary the index can itself be indexed. The other fields in the record can also be indexed but the lowest level index for each field needs to have an entry for every main record.

Records may also be processed as a 'set', such as the set of people in a department. This can be implemented by pointers which form a chain from the record for a particular department passing in turn through the records of the people in that department.

Each user of the database, whilst requiring access to data which is common to other users, can have his own program to do his own special processing. The user programs can only read or write the database by calling the central database software.

The whole database is subject to change. Within the structure outlined above, records are added or deleted as policies are taken or cancelled; a policy holder may move so that the ADDRESS field needs changing and so on. As well as all this the structure itself may change. Experience may show that an additional field is required in each record to provide more information; or perhaps a new set is to be created, containing those records which are for WHOLE-LIFE policies. It is important to keep as much of the changes as possible within the database software so that those user programs which don't need the new information need not be changed. The user programs operate on a logical data structure and the database software converts this in order to access the actual physical layout on the disc.

The interface between the user program and the database software is via the Data Description Language and the Data Manipulation Language. The currently favoured standards are those developed by the CODASYL committee and these are being implemented by most major manufacturers (except IBM). In the future we may see CODASYL give way to the relational database proposed by Ted Codd (an IBMer) which has attracted much theoretical interest.

LETTERS

MAGNETIC CORE MEMORIES

1 only Mullard AW 3307 stack containing 25 planes of 64 x 64 cores (lx,ly, 1 inhibit, 1 sense) nominal drive currents 220 mA. Max current through any wire 0.6A for 100mS. Max PD between any two wires 80V. Fitted with 8 37 way and 2 50 way Cannon plugs. Matrix plane wiring plug connections etc. in Mullard data sheets with order. £15.00

1 only Mullard type AW 3341 stack containing about twice as many cores as the one above but regret we have no official data on this one other than our own hand-written notes showing the connections to the cannon plugs. This one contains 18 Cannon plugs. £25.00

I would prefer to sell but will consider trading one for a Phillips cassette recorder and possibly exchange one for useful parts i.e. I/C etc. What have you ?? Write first or phone my home after 6 pm.

John J Smith 7 Kettlebaston Rd Leyton
London E10 7PE 01-556 3368

THE MARSDEN MACHINE

ICL 1500 installed at 20-26 Peel St, Marsden, W.Yorkshire on A62 Oldham/Huddersfield road. Any ACC member welcome to drop in & chat - just phone 56712 first.

Machine is 40K CPU, 2 x 4 deck clusters, 1000lpm printer, 1000cps paper tape and 600cpm card. Now operational !

Could members in Europe look out for RCA made Gamma 6, 10's etc that are the same as ICL 1500 as I'm looking for a disc store.

J Aslett 2 Park St Nettleton Hill Golcar Huddersfield HD7 4PB

THE GERMAN SCENE

My 12 bit machine is working well with no faults now. While in England in June I purchased a Creed 7ERP teleprinter with paper tape punch built in, and a Creed 5 unit tape reader. These are now connected to the computer and are working well. (I have modified the ASR interface to work 5 unit 50 bauds.) I managed to obtain most of the listings for the standard software (loaders, debug, assembler etc.) and have been very busy entering these

programs and punching them as object tapes. Of course I have had to modify all the tape formats for 3 times 4 bit characters per 12 bit word instead of 2 times 6 bit which was the original format with 8 channel tape. And I have added new device drivers to convert ASCII output to Baudot for the printer. I hope soon to have my tape cassette connected to the machine which will speed up the loading of programs. I have also written a number of programs myself which are running on the machine the big project at the moment is the writing of a BASIC interpreter which will permit me to run many of my games on the machine.

By the way in one of the early copies of the Newsletter (Vol 1 Iss 2) there was a 5 unit code table for teleprinters and at first sight this seems to be rather restricted especially as there isn't a plus sign or an equals sign in the character set. However this character set is the American compatible one and most of the Creed teleprinters available in the U.K. do have both plus and equals signs in the character set and as such are much more suitable as computer peripherals.

The printer I have here didn't have any cover and was pretty noisy however I built a cover mostly from hardboard with a large perspex section to view the print and the noise level is now much more acceptable.

I also purchased the RTTY handbook from the Radio Society of Great Britain and it is absolutely invaluable for setting and adjusting Creed machines. I have also purchased a desk to stand the peripherals on and its now becoming quite an acceptable installation. What I would really like added is a video display but I think that will have to wait for the moment.

I find the Weeny Bitter very interesting though I am unlikely to build it now that I have this machine however I have worked out the basics for an emulator so when you begin writing programs for it I can run them on my machine.

Ian D Spencer

LE BITTE MINISCULE

I must congratulate you for the exceptional interest of the Vol 3 issue 2 of the ACC Newsletter. As I just redesigned my own machine, I was absolutely enthusiastic about the 'weeny bitter' and would bring you some comments about the project itself and the opinions expressed in this paper.

I am afraid a £50 machine is utopic. A 256 word memory cannot be useful if you don't have the possibility to get more data or instructions from a 'mass memory' such as a cassette. But then it would cause an overflow in the CPU expense!

I don't understand tne need for a decimal display and overall, for an A/D converter (P.Madison). But, to begin the design with peripherals is to put the cart before the horse. It is preferable to begin designing the CPU peripheral interface and to work alternately on one or the other.

Since I began this letter I received your August issue. So the weeny bitter is becoming a reality! It's wonderful but do you think it is necessary to define octal and binary systems and to explain the rudiments of binary arithmetic and logic ? It would be more profitable to justify the choice of the instruction set which seems redundant regarding the data (word) manipulation and poor in loop instructions (no index register, no 'count and branch' instruction) no rotate, no bit manipulation. It looks like a poor man's PDP-8 !

I don't understand why there is a memory location reserved for the accumulator as you need anyway an actual register to do the arithmetic. Could you clarify the mechanism of I/O operations ? I am surprised by the lack of specialised instructions.

About a possible peripheral for the weeny bitter I think that even a used TTY or electric typewriter would be too expensive. Perhaps a tape reader and tape punch of the early 60's can fit into such a small budget. But, when you proposed the idea of a £50 computer, did you include I/O in the price?

In my own opinion, it is quite impossible to define a useful instruction set with one 8 bit word instruction and difficult with a 12 bit one. Don't emulate the PDP-8 ! I am convinced that its success was only due to its low price. Its instruction set is not a masterpiece! More and more instructions are necessary to build a real program. So the total number of bits used in the program is about the same as for a 16 bit machine with a more intelligent architecture. And, due to the number of PDP-8's sold, many programmers and even scientists have been marked for ever with this 'vilain canard'.

Michel DREYFUS
President de l'Assiciation Francaise des Amateurs Constructeurs d'Ordinateurs;
Villa 3 42 Rue de la Barre
95880 Enghien-les-Bains France

. . . . . .

The WB was originally conceived as a 'demonstration' machine, with very limited facilities, which would be better than normal computer hardware teaching aids and which could also be used to try out basic programming techniques . It was originally thought that it would be scrapped, and the parts salvaged, when the constructor wanted to move on to a 'real' mini. However during the design it became apparent that we could design a machine that would be capable of expansion. Hence we now have the WB-1, the £50 version, which is a subset of the full WB-2.

Nevertheless, the WB-1 can perform some useful functions, one of our members is interested in it as an I/O controller for a larger machine, another to use it to convert a 5 bit teleprinter to ASCII code.

About the level of explanation in the newsletter articles, this is a problem and I would appreciate some guidance on this.

Now for the instruction set, I really must defend this ! The 8 bit word was a basic design decision, to reduce cost for the basic WB-1. I agree that if you are designing a powerful machine from scratch a 16 bit word is better (if only because it gives you a faster machine by reducing the number of multiple word instructions). However cost not speed was the prime consideration.

An 8bit word then leads to an instruction format of;

- single word for non-memory instructions.
- single word containing the Op Code plus maybe address mode bits plus one or more additional words to give the address for memory ref instructions.

The addressing modes used came from a suggestion in a previous newsletter and fitted in nicely with our design aims as it allows us to use an 8 bit address for the WB-1 which is a natural subset of the more powerful addressing modes used in the WB2.

The choice of Op Codes was determined by the desire to keep the cost low while still giving a powerful result. In fact I am extremely pleased with the result. It is basically simple and easy to learn while being efficient in terms of memory requirements for a typical program, and it does not require too much hardware in the control logic. Comparing it with the PDP-8 there are several advantages;

- it is possible to perform unary operations on data in memory without disturbing the contents of the Acc (incedentally, this is why the Acc is a memory location; the hardware registers used in the arithmetic unit have other things to do).
- it is possible to perform binary operations on data in memory without disturbing the Acc if one of the operands is a constant (e.g. ADD #3 ABC)
- it is possible to compare two items without destroying one of them.
- there are far more data manipulation instructions.

- admittedly there are no index registers as such, but indirect addressing does the same thing.
- again there are no 'count and branch' instructions such as the PDP-8 ISZ, but the function can easily be done in software,it takes longer but simplifies the control logic. It is perhaps interesting to compare the WB & PDP-8 instruction sets for a simple loop;

```
        PDP-8                WB
        ADD COUNT            MOV  COUNTR,TEMP
        LDA TEMP
LOOP:   ....         LOOP:   ....
        ....                 ....
        ....                 ....
        ISZ TEMP             DEC TEMP
        JMP LOOP             BPL LOOP
        ....                 ....
COUNT:  COUNTR       TEMP:
TEMP:
```

note that even assuming that the PDP-8 program is within one page and we don't need to save the Acc before the ADD COUNT instruction, the PDP-8 uses 6 x 12 = 72 bits, the WB 8 x 8 = 64. A rough comparison shows that the WB uses less bits for most programs than the PDP-8, and slightly more than the PDP-11, a very powerful machine that the amateur would find very difficult to construct.

I can't agree that there are no bit manipulation instructions, you can set or clear any bit or bits (IOR and AND), or test the condition of any bit(s) (BIT) or even invert selected bits (XOR). That seems enough for a simple machine.

mike lord

ED'S BIT

Thanks to those members who have been spreading the word about the ACC. One - Ian Spencer- seems to be trying to recruit members from as many countries as possible.One new member has cast serious doubts on the viability of our plans to put the ACC membership records on a computer file by giving us his name & address in the Russian alphabet (after all, he does live there).

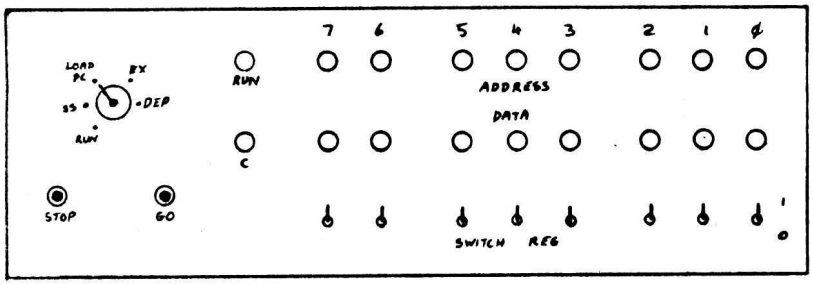
We've now got some small (A5) posters advertising the ACC so if anyone would like one to stick up in an appropriate place ! let me know.

Question of the month; what is the PASCAL language ?

Computer Weekly and DEC are running a competition for schools with a DEC computer as prize. Entry forms from Computer Weekly, Dorset House, Stamford St., London SE1 9LU. Closing data for the competition is rather close - 28 November.

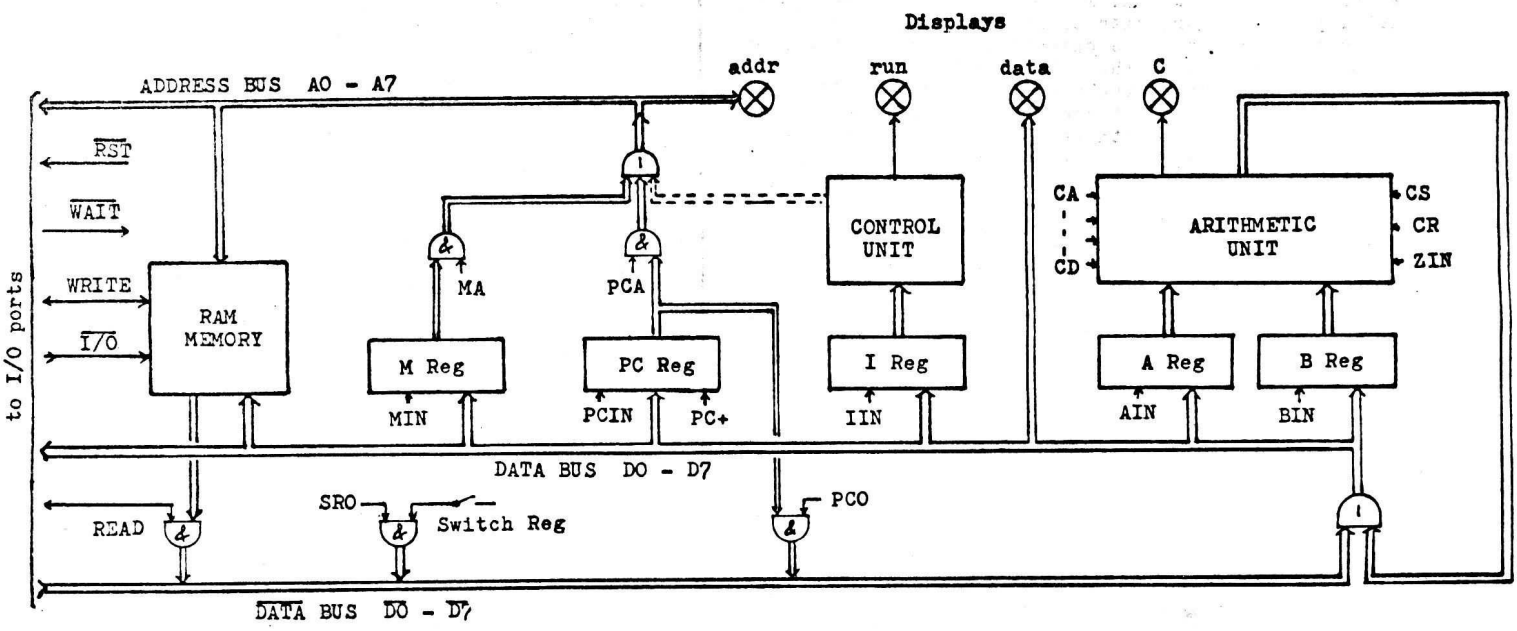
Spectronics showed a cheap display at a recent exhibition. Consists of a rotating disc , opaque except for alphanumeric characters in sequence around it. A fixed array of 15 LED are strobed at the correct times resulting in a (curved) 15 character alphanumeric display.Assosciated with it was a 6-key one handed keyboard - the inventor claims that it only takes about 20 minutes to learn to use it.

mike lord



* THE WEENY·BITTER *

HARDWARE PART I



INPUT / OUTPUT

As mentioned last time, I/O ports are treated as though they were memory locations, so if we have, say, a set of thumbwheel switches connected to an input port it would have an address such as 020. The contents of the switches could then be read into the Acc, or added to the previous contents of the Acc by instructions such as;

```
MOV  20  A
ADD  20  A
```

Similarly data could be sent from the Acc to an output port at location 022 using the instruction;

```
MOV  A  22
```

Note that by definition an input port sends data to the CPU and an output port receives data from the CPU only, an attempt to read data from an output port into the CPU will give a result of zero. This is particularly interesting in the case of the single operand memory reference instructions e.g. INC X as they first get the data from the memory location, then modify it, and finally store the result. As they will get zero from an output port, the instruction;

```
INC  22
```

will result in the number 001 being sent to the OP port @ loc 22.

Although possible, it is not advisable to give the same address to both an input and an output port.

Hardware Interface

All I/O ports connect onto a common set of data buses (see block diagram). Up to 20. ports can be connected without adding any bufferring if the following rules are observed;

- the bus wiring should be as short as possible, preferably less than 2' overall.
- solid, short, earth (CV) connections must be provided between the ports and the CPU.
- each port must present not more than one standard TTL load to the Address and Data buses and the WRITE & READ control lines.
- outputs from the ports onto the Data bus and the WAIT & I/O lines must be via a standard open collector TTL gate such as the 7401.

Address Bus lines A0 - A7 are normal levels (high = 1) , A0 is the least significant bit. As the expanded version of the WB can have up to 16. address lines, an extra line (AL0) is provided which is '1' (High) when the address present is in the range 0 - 255. Although not necessary on the WB-1, it is probably worth providing for this input on any I/O port.

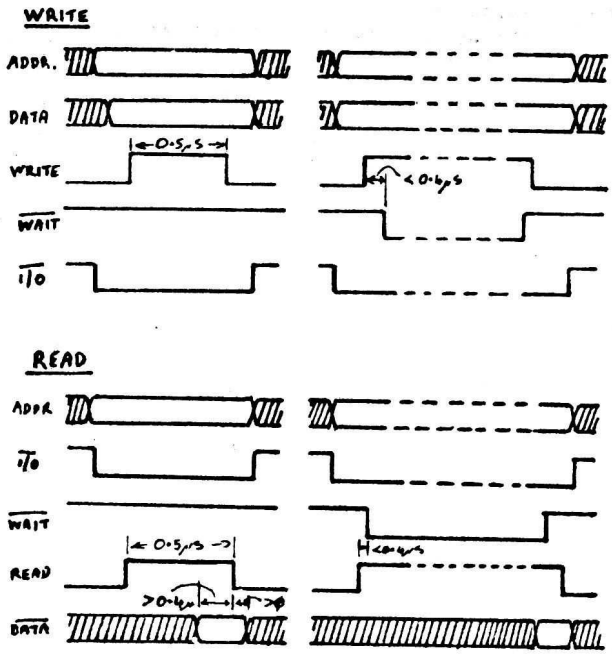
The READ line is brought high (to 1) by the CPU when it wants to get data from the memory or I/O. It is only at 1 when the address information is valid. It stays at '1' for at least 0.5uS.

The WRITE line is brought high (1) by the CPU when it wants to send data to memory or I/O. It only goes to '1' when both the address and the data are valid. It stays at '1' for at least 0.5uS.

The I/O line must be pulled down by an I/O port when it recognises its own address - and it must remain at 0 for as long as that address is present, regardless of whether the CPU is performing a write or read, and of whether the I/O port is for input or output.

If the I/O device (or memory) takes longer than 0.5uS to transfer data it can suspend the CPU operation by pulling the WAIT line down to 0. The CPU then pauses, remaining in the write or read state, until the WAIT line is allowed to go high again.

The RST line is pulled down low by the CPU for about 1mS when power is first applied, and is to be used to reset the peripherals to the start-up condition



Protocol

There are three basic ways of effecting an input or output transfer;

- by using the WAIT line to stop the CPU until the peripheral has finished.Thus for output to, say, a punch, the command

```
MOV A PUNCH
```

would transfer the contents of the Acc to the punch but the output port circuits would pull the WAIT line down, stalling the CPU, if the punch hadn't finished with the previous character. Similarly, the command

```
MOV READER A
```

could cause the paper tape reader to read the next character from tape, stalling the CPU until it had been read in.
This technique is simple but suffers from the disadvantage that the CPU can't do any processing at all when it is waiting for the peripheral.

- by arranging for the peripheral to 'interrupt' the CPU when it is ready.This is probably the most flexible arrangement and is catered for on the enhanced WB. (An additional INT line is provided between the CPU and I/O ports; and it is pulled down to 0 by the interrupting device).

- by adding'status registers' to the I/O ports. These status registers are additional I/O ports, but they handle peripheral device control rather than data.Examples are given below for a TTY keyboard/reader and printer/punch.

TTY Keyboard/Reader

Two I/O ports are used;
PRB is a read-only data port.
PRS is the status register and it uses the following bits;

- B7 (read only)is set to one when a character has been received from the TTY and is present in PRB.It is cleared when PRB is accessed by the CPU.
- B0 (write only) enables the reader tc read one character from paper tape when it is set to one by the CPU.
- B6 is reserved for an 'interrupt enable' function.
- B5 (read only) is set to one when an error occurs e.g. end of tape.

The following program shows how the CPU can interleave other work with reading a character from tape;

```
        INC  PRS    (enable reader)
        ...         (other work)
        ...
LOOP:   TST PRS     (see if char has been read)
        GPL LOOP    (hang about)
        MOV PRB A   (put character into Acc)
```

TTY Printer/Punch

PPB is the write only punch data buffer, writing a character into PPB operates the punch.

PPS is the status reg;

B7 (read only) is set to one when the punch is ready to accept another character.

B6 is reserved for the 'interrupt enable'.

B5 (read only) is set to one when an error occurs

The following program will copy a paper tape from reader to punch ;

```
LOOP:   INC PRS     (enable reader)
WAITR:  TST PRS     (wait for rdr)
        BPL WAITR
WAITP:  TST PPS     (wait for punch)
        BPL WAITP
        MOV PRB A   (transfer char)
        MOV A PPB   (punch it)
        GTO LOOP    (and back again)
```

HARDWARE

A hardware block diagram is given on page 8, showing the major data and control paths and units. Main differences between this and the programmers' view of the machine (see last issue) are;

The M reg, used to hold the address of the data in memory ref instructions such as TST X

The Instruction reg and control unit.

Division of the Data Bus into two parts; the $\overline{\text{DATA}}$ Bus which is a 'wired-or' connection which gathers the outputs from memory, switch reg, I/O etc., and the DATA bus which is a buffered version of the $\overline{\text{DATA}}$ bus 'or-ed' with the output of the Arithmetic Unit.

The WB-1 is being built at this very moment!, but is not yet complete. For this reason (and lack of space) details of the control unit are being left until the next issue - they are quite complex & we want to be sure that everything is OK before going into print. Schematics of the other unitsare included in this issue. We haven't made any attempt to produce printed circuit board layouts as they would be extremely complex and difficult (= costly) to make.

The WB has ended up using rather a lot of I/C packages - mainly to reduce the component cost. In many places complex LSI I/C could have been used to reduce the package count but this would have been a rather expensive approach for the amateur.

The WB is best built onto 3 boards; Arithmetic unit, address logic and control logic. Each board needs to be about 8" x 8" and, if you are going to use plug/sockets for connection (the boards could be hard-wired together) we want at least 58 connections to each board. I find the best approach is to use plain (no copper) board pierced at 0.1" intervals. Either thick wire or copper foil strip should be used in a mesh layout for 0v and +5V power distribution on each board, signal wiring by thin insulated wire ('self fluxing' enamel copper wire is ideal if you can get hold of it). Low impedance power supply connections to each IC are most important if the machine is to perform reliably; although not shown on the schematics, a small (0.1 to 0.01uF) capacitor should be connected between 0V & +5V at every other IC and an electrolytic (5 - 100uF) used where the power comes in to the board. Total power requirements for the WB-1 are about 1.5A at 5V.

NOTES ON SCHEMATICS

Numbers shown just outside the logic elements are the package pin numbers.

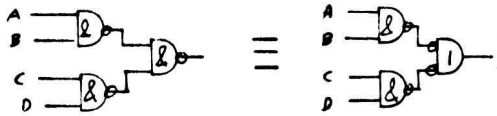
A logical '1' is 'high' (2.4 to 5V)

A logical '0' is 'low' (0 to 0.8V)

We have used the trick of putting a bar ($\overline{\ }$) above a signal name if the LOGICAL LEVEL is inverted with respect to the INFORMATION LEVEL at that point. For example, consider bit 6 of a data word. When the word is present on the DATA Bus then if the bit is a '1' (INFORMATION LEVEL = 1) then the line D6 will be high (LOGICAL LEVEL = 1). On the $\overline{\text{DATA}}$ Bus, however,the line ($\overline{\text{D6}}$) would be low (LOGICAL LEVEL = 0). The bar reminds us of this.

Similarly, we have taken advantage of the fact that most gates have a dual function and so can be drawn in two ways. e.g. the 7400 can be considered as a NAND gate (high = 1) or a NOR gate (low = 1)

This can simplify comprehension of the circuit function. The two circuits shown below are in fact identical but the overall function (A and B or C and D) is more readily understood from the right-hand version.(At least, that's my belief).



WEENY-BITTER COMPONENTS

Arithmetic unit

2 x 74181 X1,2	3 x 7404 X12,15,16
4 x 7475 X3,4,10,11	2 x 7401 X13,14
2 x 7474 X19,20	2 x 7437 X17,18
1 x 7400 X5	6 x 7430 X21-26
4 x 7451 X6,7,8,9	1 x 74154 X27
9 x 1K resistors	9 x 180 ohm resistors
8 x 390 ohm resistors	

Address circuits & memory

2 x 2101 X1,2	1 x 7408 X11
2 x 7475 X3,4	1 x 7437 X17
2 x 74193 X 5,6	1 x 7405 X14
4 x 7400 X7 - 10	2 x 7404 X18,19
4 x 7401 X 12,13,15,16	2 x 7440 X20,21
8 x 180 ohm resistors	1 x 390 ohm resistor

Front Panel

18 x LED
8 single pole single throw toggle switches
2 single pole change-over push buttons
1 single pole 5 way rotary switch

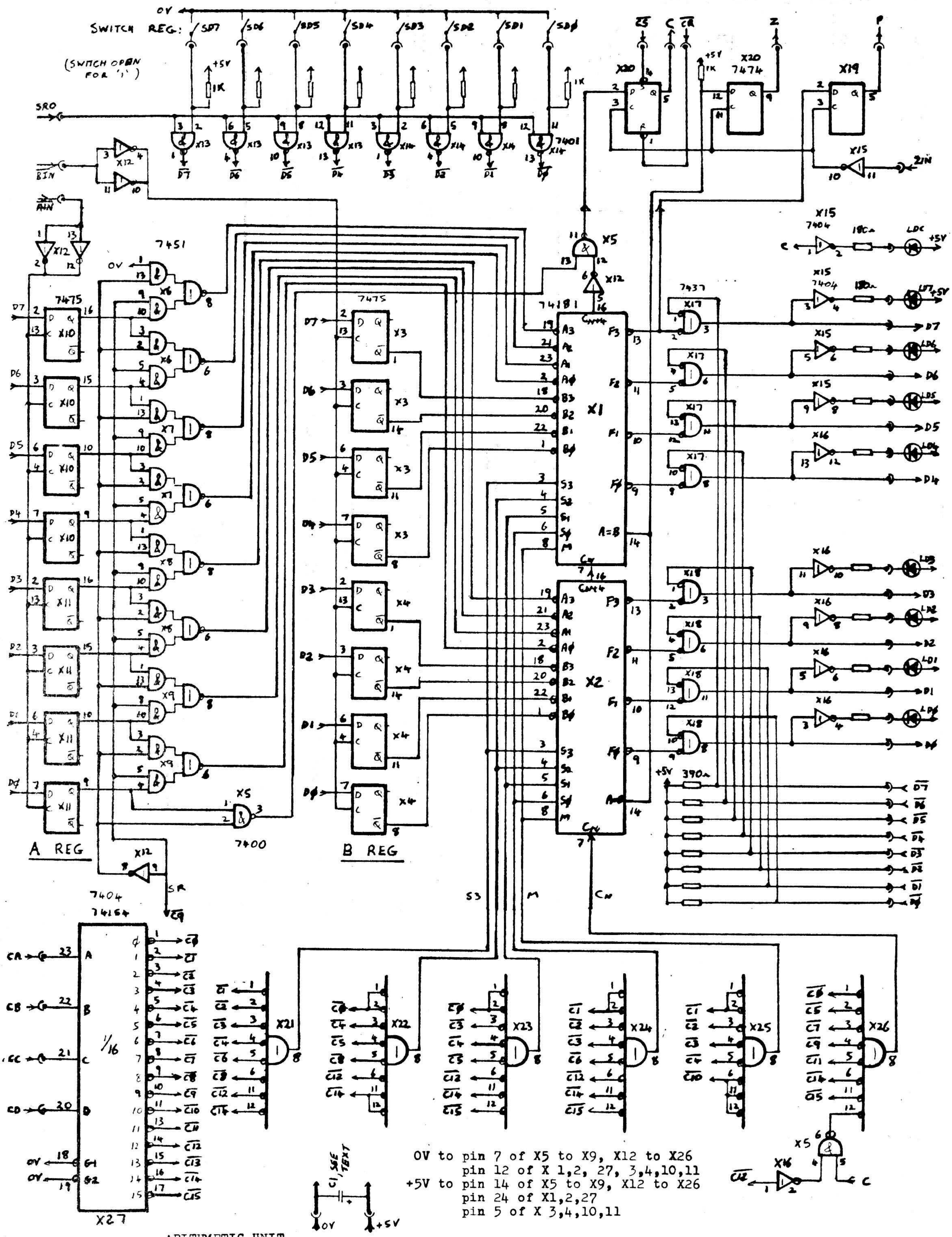
Control circuits (approx - see text)

6 x 7474	2 x 7475
5 x 7408	1 x 7437
8 x 7400	3 x 7420
7 x 7410	1 x 7405
3 x 7404	1 x 7430
	1 x 7413
7 x 1K resistors	1 x 180 ohm resistor

Arithmetic Unit

This comprises the A & B registers (X10,11 and X3,4), two 74181's used to perform most of the data manipulation, a gate array (X6,7,8,9) which performs the Right Shift operation,the OR gates which also act as buffers to drive the Data Bus (X17,18) and the Z,C & P registers (X19,20). This board also carries the switch register gates (X13, 14) and data display drivers (X15,16).

X21 to 27 take four control inputs (CA - CD) and set up various control inputs to the shift gates and the 74181's accordingly. They could be replaced by a 16 word, 7 bit ROM. The coding of the CA - CD lines corresponds to the relevant portion of the WB instruction set (see table).



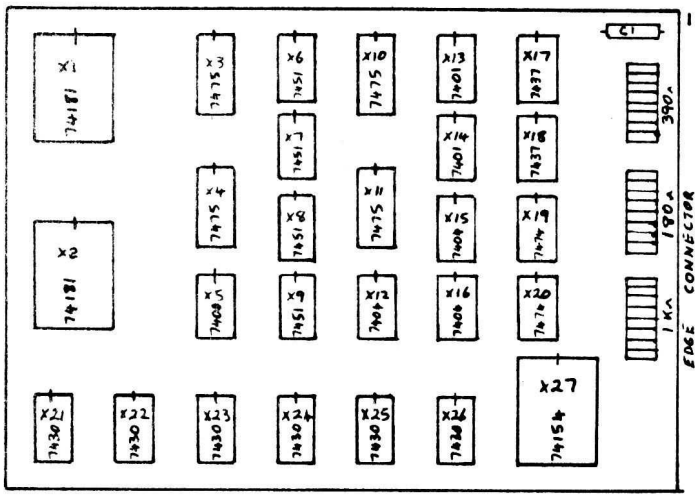
0V to pin 7 of X5 to X9, X12 to X26
pin 12 of X 1,2, 27, 3,4,10,11
+5V to pin 14 of X5 to X9, X12 to X26
pin 24 of X1,2,27
pin 5 of X 3,4,10,11

ARITHMETIC UNIT

Coding of AU control inputs CA-CD

inputs				operation							
CA	CB	CC	CD		SR	S3	S2	S1	S0	M	Cn
0	0	0	0	A minus B	0	0	1	1	0	0	1
1	0	0	0	A XOR B	0	1	0	0	1	1	0
0	1	0	0	A XOR B	0	1	0	0	1	1	0
1	1	0	0	A IOR B	0	1	0	1	1	1	0
0	0	1	0	A AND B	0	1	1	1	0	1	0
1	0	1	0	A minus B	0	0	1	1	0	0	1
0	1	1	0	A plus B	0	1	0	0	1	0	0
1	1	1	0	A	0	0	0	0	0	0	1
0	0	0	1	A shift L	0	1	1	0	0	0	0
1	0	0	1	A shift R	1	0	0	0	0	0	1
0	1	0	1	$\overline{A}$	0	0	0	0	0	1	0
1	1	0	1	A	0	0	0	0	0	0	1
0	0	1	1	A plus C	0	1	1	1	1	0	C
1	0	1	1	A minus 1	0	0	0	0	0	0	0
0	1	1	1	A plus 1	0	1	1	1	1	0	1
1	1	1	1	zero	0	0	0	1	1	0	1

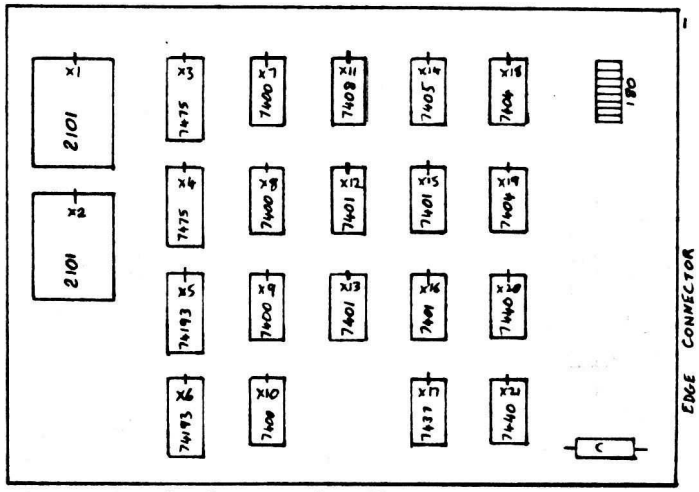
The unit can be tested by itself as follows; first check the operation of X21 - 27 according to the table above.
Then tie CA-CD to +5V (1) and check for high levels from the F outputs of X1,2. With SRO at OV (0) lamps LD0-LD7 should be out and the Data Bus lines $\overline{D0}$-$\overline{D7}$ should be low. Connecting any of the lines $\overline{D0}$-$\overline{D7}$ to OV should light the corresponding lamp and make the corresponding D line go high. The Switch Reg gates can now be checked by connecting SRO to +5V, enabling gates X13,14.
Next, connect $\overline{AIN}$ & $\overline{BIN}$ to +5V through 1K resistors. The signal on the Data Bus (controlled by the switch reg when SRO is high) can now be gated into the A or B reg by momentarily connectint $\overline{AIN}$ or $\overline{BIN}$ to OV. Operation of the 74181's and the shift-right gates can now be checked by setting appropriate patterns into the A & B regs from the switch reg then inhibiting the switch reg by connecting SRO to OV and setting up the required code on CA-CD.
Finally connect ZIN to +5V via 1K then check the operation of the C,Z & P reg by momentarily connecting ZIN to OV.



ARITHMETIC UNIT, A & B REG
PROPOSED LAYOUT

ADDRESS LOGIC AND MEMORY (see page 14)

This comprises the M reg (X3,4) Program Counter (X5,6), the Memory itself (X1,2), Address Bus drivers (X17,20,21), Address Display drivers (X18, 19) and miscellaneous gating circuits.
Information from the Data Bus can be gated into either the M or PC reg by momentarily pulling $\overline{MIN}$ or $\overline{PCIN}$ low.
Information put onto the Address Bus can be from the M reg, or the PC reg (by making MA or PCA high) or neither, in which case address lines A3 to A7 are low (0) and the state of the three low order address lines A0 to A2 can be controlled by the control unit via lines $\overline{IA0}$ - $\overline{IA2}$.
The open-collector gates X12,13 transfer the contents of the Program Counter onto the Data Bus when PC0 is high. $\overline{PC+}$ is used to increment the Program Counter (count up). The count-down ability of the 74193's is not used.
Open collector inverters X14 are used to make a 6 input negative NAND gate giving an output AX which goes high when A2 to A7 are all at 0 . This signal is used by the control circuits.
The INTEL 2101 256 word by 4 bit RAM was chosen as it has separate inputs and outputs. (the 2111 or 2112 could be used if some extra gates were added). It is normally left in the READ mode and it's outputs gated onto the $\overline{Data}$ Bus via X15,16 when required. It is available from Rapid Recall Ltd., 9 Betterton St., London WC2H 9BS as the P2101 at £3.04 plus VAT, P&P.



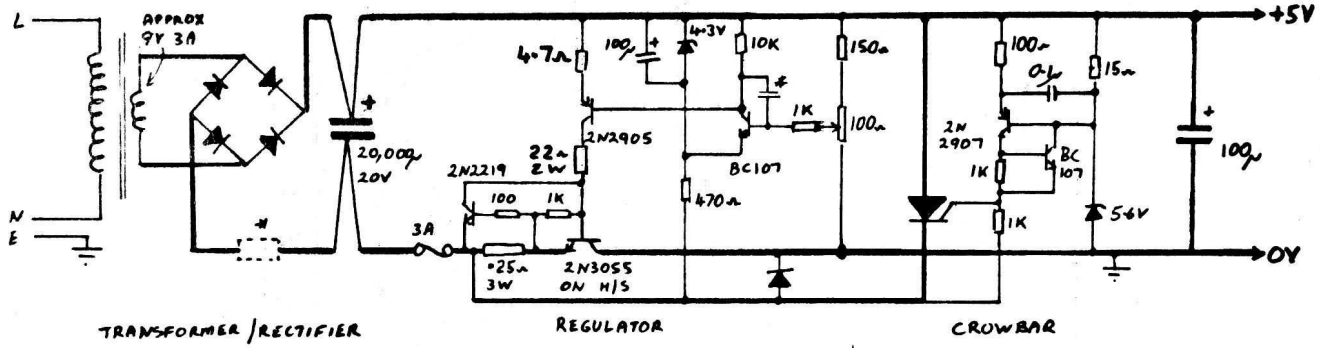
ADDRESS CIRCUITS + MEMORY
PROPOSED LAYOUT

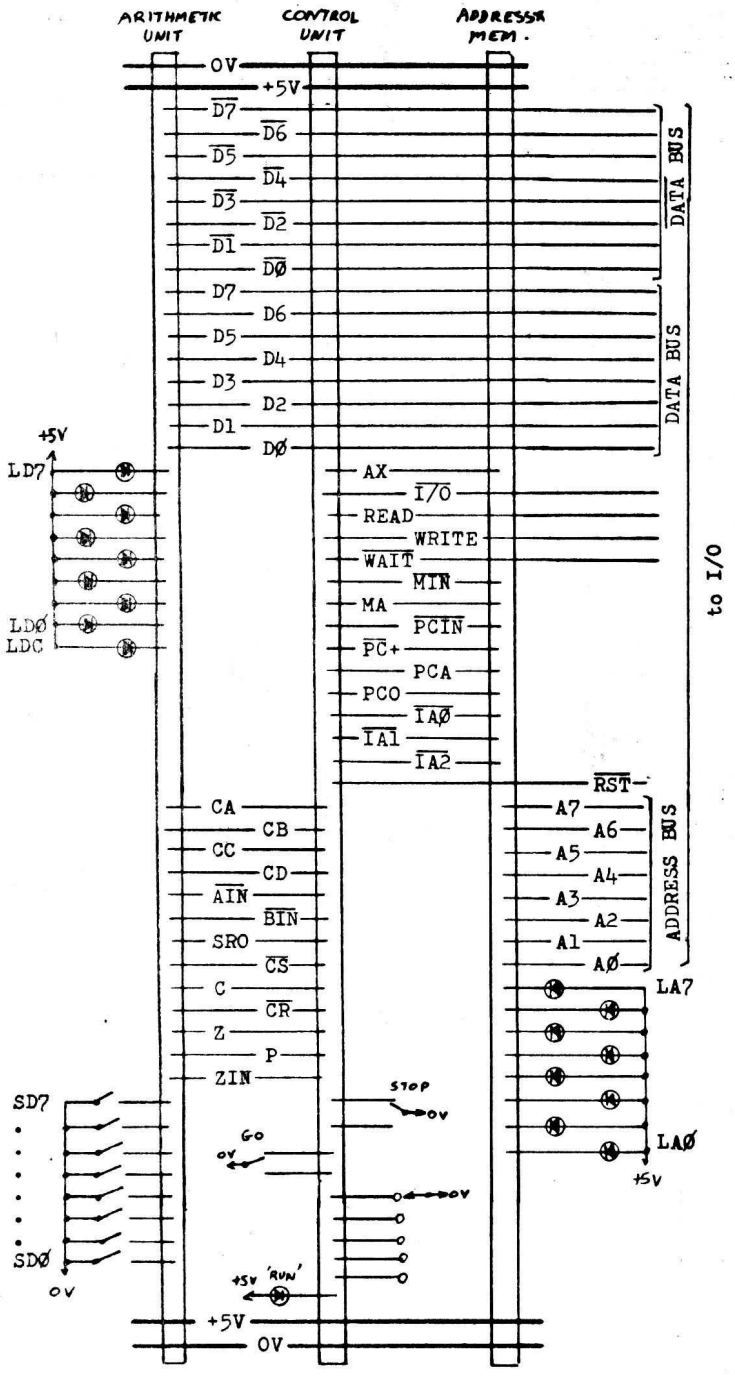
POWER UNIT

The unit shown on page 13 will supply about 2.5A at 5V; sufficient for the basic WB plus a few peripheral circuit ports. The five diodes can be any rectifier capable of handling 3A , 50V. The regulator circuit shown provides current limit protection at about 3A current. The 2N3055 can dissipate up to 20W under fault conditions so should be mounted on a fairly substantial heatsink (its collector is a earth potential so can be mounted directly onto a chassis without insulation.). If it is preferred the regulator could be replaced by two or more paralleled I/C 5V regulators. The capacitor marked with a * may be necessary to stop oscillation of the regulator and should be between 100pF and 0.1uF. A resistor may be fitted in series with the rectifier bridge as shown * to reduce the dissipation in the regulator.
The crowbar is a most worthwhile insurance against accidental overvoltage on the 5V supply, either from failure of the regulator or because of accidental connection of an external voltage. It can be tested by replacing the 3A fuse by a 10 ohm 20W resistor, disconnecting the output of the supply from any load, then increasing the output voltage until the SCR fires. This should happen at between 5.5 and 7 Volts. The SCR can be any device rated at better than 5A.

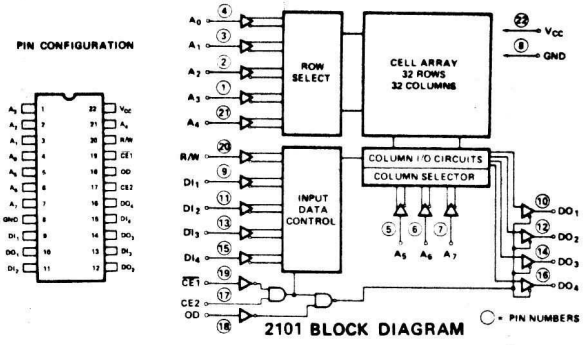
INTERCONNECTION OF THE UNITS

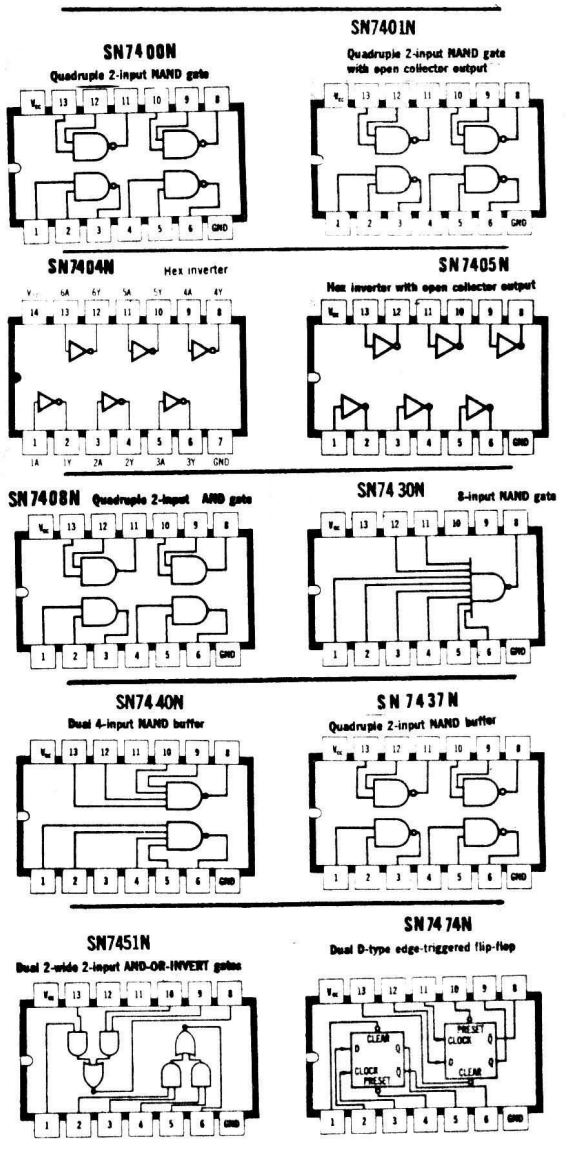
The necessary wiring between the three units and the lamps and switches on the front panel is shown on page 13. Note the use of at least two connections for each of the power supply lines OV & +5V

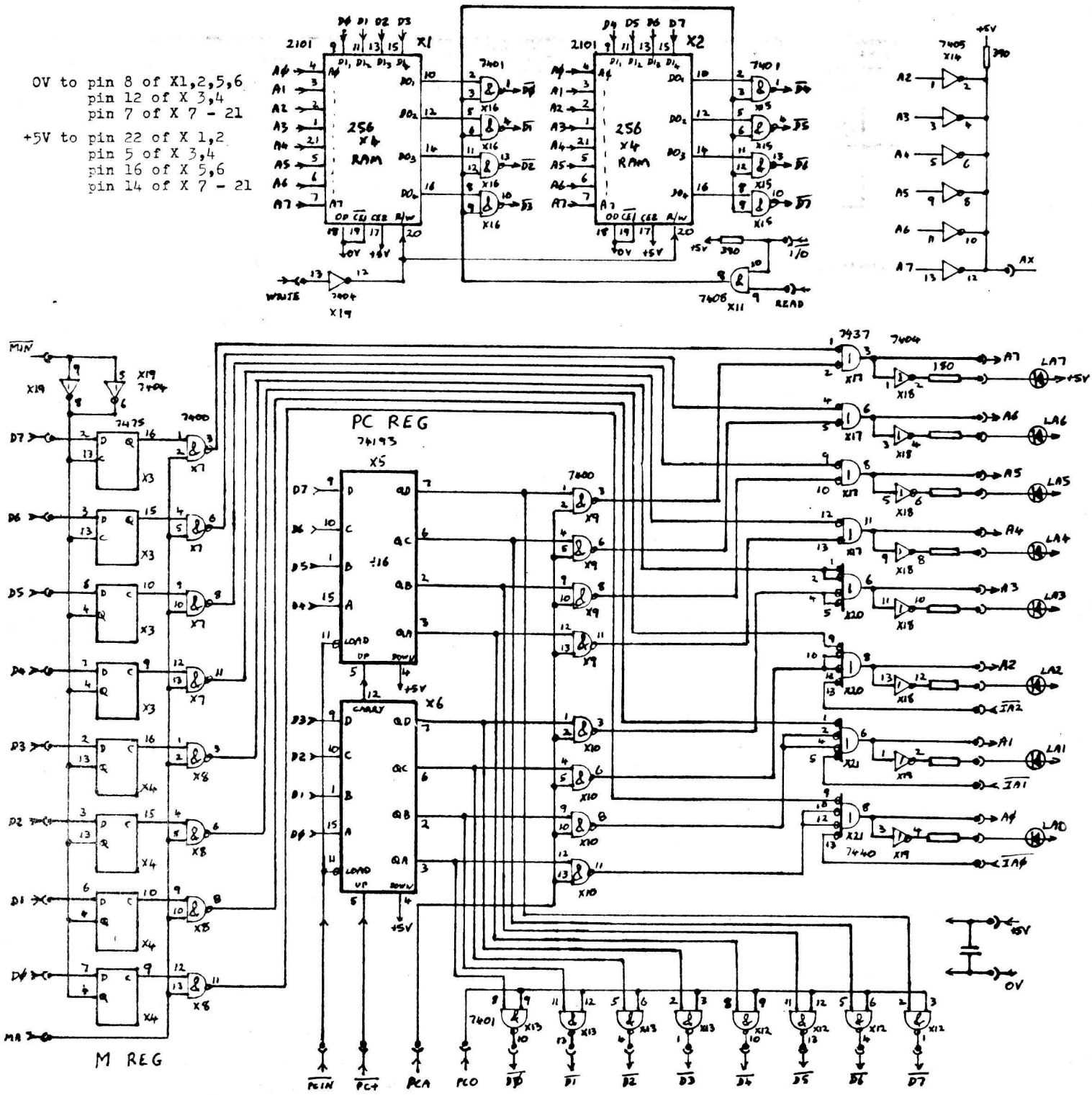




CONNECTIONS BETWEEN UNITS

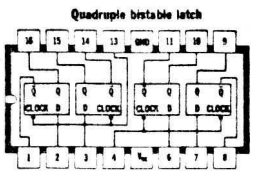






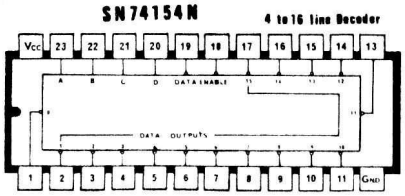
SN7475N

Quadruple bistable latch



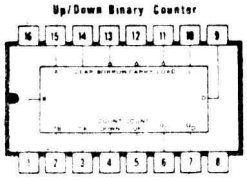
SN74154N

4 to 16 line Decoder



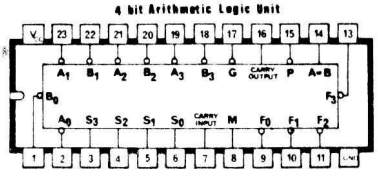
SN74193N

Up/Down Binary Counter



SN74181N

4 bit Arithmetic Logic Unit



AMATEUR COMPUTER CLUB NEWSLETTER
Vol 3 Iss 4 October '75

m.lord
7 Dordells, Basildon, Essex
tel; 0268 411125 (home)
0268 3040 x 117 (work)