```
1 REM ****************************************
2 REM *  HERE FIRST THE SOFT-          *
3 REM *  SWITCHES / PEEKS AND          *
4 REM *  POKES USED WITH THE           *
5 REM *  GAMEPORT – THIS BLOCK         *
6 REM *  MAY BE REMOVED TO KEEP        *
7 REM * PROGRAMCODE COMPACT            *
8 REM *      R = READ      W= WRITE     *
9 REM *      R/W = READ AND WRITE       *
10 REM * ALL ADRESSES ARE MARKED *
11 REM * AS HEXADECIMAL ADRESSES *
12 REM * $C040 = UTILITY STROBE   R    *
13 REM * $C058 =  AN0 = LOW          R/W *
14 REM * $C059 =  AN0 = HIGH         R/W *
15 REM * $C05A =  AN1 = LOW          R/W *
16 REM * $C05B =  AN1 = HIGH         R/W *
17 REM * $C05C =  AN2 = LOW          R/W *
18 REM * $C05D =  AN2 = HIGH         R/W *
19 REM * $C05E =  AN3 = LOW          R/W *
20 REM * $C05F =  AN3 = HIGH         R/W *
21 REM ****************************************
22 REM * HERE ARE THE PUSHBUTTON*
23 REM * ADRESSES WHICH WE WILL   *
24 REM * USE AS INPUT DETECTORS    *
25 REM ****************************************
26 REM * $C061 = PB0                      R *
27 REM * $C062 = PB1                      R *
28 REM * $C063 = PB2                      R *
29 REM * $C064 = PADDLE 1 - X AXIS  R *
30 REM * $C065 = PADDLE 1 - Y AXIS  R *
31 REM * $C066 = PADDLE 2 - X AXIS  R *
32 REM * $C067 = PADDLE 2 - Y AXIS  R *
33 REM ****************************************
34 REM * HERE ARE THE ADRESSES      *
35 REM * LISTED ALSO AS DECIMALS    *
36 REM * STROBE% = 49216       $C040  *
37 REM * AN0% = 49240 = LOW    $C058  *
38 REM * AN0% = 49241 = HIGH   $C059  *
39 REM * AN1% = 49242 = LOW    $C05A  *
40 REM * AN1% = 49243 = HIGH   $C05B  *
41 REM * AN2% = 49244 = LOW    $C05C  *
42 REM * AN2% = 49245 = HIGH   $C05D  *
43 REM * AN3% = 49246 = LOW    $C05E  *
44 REM * AN3% = 49247 = HIGH   $C05F  *
45 REM ****************************************
46 REM *   PB0%  =  49249 =  $C061         *
47 REM *   PB1%  =  49250 =  $C062         *
48 REM *   PB2%  =  49251 =  $C063         *
49 REM *   P1X%  =  49252 =  $C064         *
50 REM *   P1Y%  =  49253 =  $C065         *
51 REM *   P2X%  =  49254 =  $C066         *
52 REM *   P2Y%  =  49255 =  $C067         *
53 REM ****************************************
```

So up to this point you may deny to type in the code – it´s just a comment on used values…. At the other hand… if you type it in and save it with a title like “gameportheader” you might use a utility like the “merge” command from the Beagle Brothers and use it similar to other parts of the program and keep by that your program well documented – and by this habit it is more comfortable to alter parts of the program anytime later…

To make programs readable and usefull it is wise to use instead of the numbers variables and give that variables the correct numbers…

So in the next part we setup the variables – **this has to be done before you use them** – otherwise the program reads a variablename that has not been defined yet and you can´t predict the content given to it by the program !

```
54 REM ****************************************
55 REM * DEFINE VARIABLES                *
56 REM ****************************************
57 LET STROBE% = 49216
58 ************************************************
59 LET AN0OFF% = 49240
60 LET AN0ON% = 49241
61 LET AN1OFF% = 49242
62 LET AN1ON% = 49243
63 LET AN2OFF% = 49244
64 LET AN2ON% = 49245
65 LET AN3OFF% = 49246
66 LET AN3ON% = 49247
67 REM ****************************************
68 LET PB0% = 49249
69 LET PB1% = 49250
70 LET PB2% = 49251
71 REM ****************************************
72 LET P1X% = 49252
73 LET P1Y% = 49253
74 LET P2X% = 49254
75 LET P2Y% = 49255
76 REM ****************************************
```

Just a short experiment here at this point for example:
The address-switch for the speaker is 49200 so the following commandline :

```
100 A = PEEK(49200)
```

generates a short click at the loudspeaker.
By giving the variable A the value of the address-content from 49200 at the very moment of the execution of this line generates a reading-access to that address and that causes the click-noise.
So in the following program a PEEK-command generates a reading-access to a given address that is specified in the brackets after the PEEK –command and a POKE-command generates

a write-command to a given adress with the same syntax … so:

POKE 49246,1

would write a “1” in the given adress. Note also: Variables that contain at the end a “%”-sign contain a “normal” number as value !

But be aware of the content in the header – it shows the switches that are “read-only” by the marking at the end of the line with only a “**R**”. Such adresses are only permitted to be used with the “PEEK”-command. A “POKE”-command would cause a “break/stop” in the program and probably a error-code.

Another important point at this stage - to be explained are the accesses to the Paddle / Joystick – switches (i.e. line **49** to line **52** in the header of page 1). That are “analog” inputs and they might become any value from 0 to 255.
A value above 127 is a move forward and a value equal or less 127 is a move backward with the joystick and in horizontal move it´s similar - above 127 is a move to right side and equal or less is a move to the left.

Basic specifics at the pushbuttons PB0 to PB3 is similar: a value above 127 means the Button is pressed and a value equal or less means that the Button is released.

At the very moment the joystick-switches are without use in this first program, but they will be used later within other programs and experiments.

The PCB is designed for different kind of use and at this point it is necessary to decide which way you want to use the switches:

a) you might only want one of 16 switches to work in a time and the others to be switched “off” – that will be similar to a “spot-bargraph” and in that case the pole of the output from the 74LS154 are connected from the 16-pol connector to the connectors of the bases of the transistors “line by line”

or

b) you want to use switches at the same time - but then you might only use 4 switches. In this case the pins of **AN0** to **AN3** from the 15 pol. connectors are used and 4 lines are connected to the 4 selected pins of the base of the selected transistors.

Take a look at the drawing after this program to see both alternate choices.
In case **a)** we use the decoding of the 74LS154 in the case **b)** we don´t.

As explained in previous part a) the adress is decoded by the 74LS154 and in b) the lines from **AN0** to **AN3** are used directly for switching – but mixing both modes for example is possible with minor restrictions by using **AN0** to **AN2** by “mixing-mode” **and** if AN3 is “off” the switches from 9 to 16 are all “off” **or** if **AN3** is “on” **<u>one switch</u>** of the switches from 9 to 16 is set “on” as “1 out of 8”, depending to the setting of **AN0** to **AN2**.

So using the AN0 to AN3 switches the following table shows the use of the bits as code in a binary system where
VaAN0% represents **0** or **1** (i.e. Bit 0 or 2^0) ,
VaAN1% represents **0** or **2** (i.e. Bit 1 or 2^1),
VaAN2% represents **0** or **4** (i.e. Bit 2 or 2^2) and
VaAN3% represents **0** or **8** (i.e. Bit 3 or 2^3).

VaAN0%	VaAN1%	VaAN2%	VaAN3%	Value
0	0	0	0	0
1	0	0	0	1
0	1	0	0	2
1	1	0	0	3
0	0	1	0	4
1	0	1	0	5
0	1	1	0	6
1	1	1	0	7
0	0	0	1	8
1	0	0	1	9
0	1	0	1	10
1	1	0	1	11
0	0	1	1	12
1	0	1	1	13
0	1	1	1	14
1	1	1	1	15

Just for example lets view the value of **10** by examining the row so the value results from the addition of those numbers with “**1**” set (i.e. they are “**on**” so **AN1** =”**2**” together with **AN3** = “**8**” and the sum is 8 + 2 = 10.

So by setting the **AN0** to **AN3** in the case of **a)** you can determine by the decoding of the 74LS154 any Unit from **0** to **15** (or in normal language 1 to 16) to switch to “**ON**” while the others are switched “**OFF**”.

In case **b)** you can set any combination within the columns **AN0** to **AN3** !

So within the next page we´ll start the “math” within the program.

Beware when performing “math” within the task of programming ! there is a difference between “counting” (i.e. 1, 2, 3, 4, and so on) and the usage of numbers on a computer… (where you must start counting **0**, 1, 2, 3, 4, And so on) – just look at the table on the previous page and you will see the difference !

Math on a computer is math with so called “digits” and working with “octal” or hexadecimal-system where 16 “bit”-values are represented with one hexadecimal-“digit” where 2 of this “hexadecimal-digits” represent one so called “byte” and that byte stores 8 single bits and therefor 0 to 255 values (i.e. in “normal” numbers 256 values !). Counting with hexadecimal-system goes like this: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F ! So “normal” numbers are regular **1 value less** if used in hexadecimal system.

In normal life this could be ignored – **but in programming a computer this makes a big difference and is very important !**

Determining the single bits out of a number is quite similar to the use of a sieve in real life to split different sizes of stones from each other…. – you must work from larger down to smaller ones….

So in this example getting down from 15 to 0 first question:
is the number larger than “7” – if “yes” then subtract **8** and set the value **VaAN3%** to “**1**” and if not the value of **VaAN3%** must be “**0**” and
in the next step the question will be:
- is the remaining rest amount larger than “3” – if “yes” then subtract **4** and set the value of **VaAN2%** to “**1**” otherwise the value of **VaAN2%** must be “**0**” and the next question has to be:
– is the remaining amount larger than “**1**” – if that is true you must subtract **2** from that amount and set the value of **VaAN1%** to “**1**” and if its not true the value of **VaAN1%** must be “**0**” and the very last question will be:
- is the remaining amount greater than “0” – if “yes” **VaAN0%** must be set to “**1**” otherwise **VaAN0%** must have a value of “**0**” !

In a program the values of **VaAN0%** to **VaAN3%** should be set alltogether to “**0**” which ensures that each time the routine is started the variables are in a defined condition …

– and **recognize the difference** between **AN0%** and **VaAN0%** – that are 2 different variables !
AN0% stores the value of the memoryadress and
VaAN0% stores the value of the BIT !

Lets take case a) and lets permit the value to be picked with an input and its stored in the variable called “choice%”

So lets view the code:

```
77 REM ****************************************
78 REM *  Lets first instruct the user about   *
79 REM *  the input-options on the screen      *
80 REM ****************************************
81 PRINT “ TYP IN A NUMBER FROM 0 TO
16 AND HIT THEN ENTER”
82 PRINT “ 0  QUITS THE PROGRAM”
82 PRINT “ 1  SWITCHES RELAY 1 ON =
LIGHTBULB RED “
83 PRINT “ 2 SWITCHES RELAY 2 ON =
LIGHTBULB ORANGE “
84 PRINT “ 3 SWITCHES RELAY 4 ON =
LIGHTBULB YELLOW “
85 PRINT “ 4 SWITCHES RELAY 5 ON =
LIGHTBULB GREEN “
86 PRINT “ 5 SWITCHES RELAY 6 ON =
LIGHTBULB VIOLETT “
87 PRINT “ 7 SWITCHES RELAY 7 ON =
LIGHTBULB BLUE “
88 PRINT “ 8 SWITCHES RELAY 8 ON =
STROBOSKOPE “
89 PRINT “ 9 SWITCHES RELAY 9 ON =
BASSHORN “
90 PRINT “ 10 SWITCHES RELAY 10 ON =
MIDHORN “
91 PRINT “ 11 SWITCHES RELAY 11 ON =
TWEETHORN “
92 PRINT “ 12 SWITCHES RELAY 12 ON =
SIRENE “
93 PRINT “ 13 SWITCHES RELAY 13 ON =
POWERCORD CAMERA “
94 PRINT “ 14 SWITCHES RELAY 14 ON =
RADIO “
95 PRINT “ 15 SWITCHES RELAY 15 ON =
CASSETEPLAYER BARKING DOG “
96 PRINT “ 16 SWITCHES RELAY 16 ON =
POWERCORD ELECTRIC MAGNET”
97 PRINT “ PLEASE CHOOSE DEVICE “
99 INPUT CHOICE%
```

Its usually a good programming behaviour – if the progam is used by people that are not familiar with the program to use “CHOICE”

instead of “CHOICE%” and therby permit to input everything and thereafter to have a programroutine (encapsuled in a GOTO-loop or a GOSUB-loop) and that routine to filter out permitted inputs and passover then to the program the values

or if input is not-permitted to display a note at the screen with information on the error and turn back to input ….

If you don´t do that - you take risk that a not permitted input causes a break in the program - and a possible damage to connected devices… this filtering techniques should be discussed in a later other programming example more in detail for different kinds of input …..
Here now only a very short check of the input:

```
100 REM ** SHORT INPUTCHECK **
101 IF  CHOICE% < 17 THEN GOTO 120
100 IF CHOICE% > 16 THEN PRINT “ THE NUMBER IS TOO LARGE “
109 REM * WRONG NUMBER GO BACK TO INPUT *
110 GOTO 81
114 REM ** LINE 2000 IS PROGRAM END **
115 IF CHOICE% = 0 GOTO 2000
118 REM *INPUT OK GO AHEAD TO SIEVE*
```

You could have probably inserted above a “HOME” command but this will delete the hint that the number was too large…. So if you want the “HOME” command you should insert a line like “99 PRINT ERROR$” and set between line 101 and line 108 something like this: 101 IF CHOICE% > 16 THEN ERROR$ = “THE NUMBER IS TO LARGE” , 102 IF CHOICE <17 THEN ERROR$ = “ “ and 103 HOME and use them instead of line 100.

Now lets do the work making the discussed sieve….

```
119 REM ***  ENTRYPOINT OF SIEVE AND TURN NUMBER TO HEXVALUE ***
120 CHOICE% = CHOICE% -1
122 REM * LETS PICKUP HIGHEST BIT *
123 IF CHOICE% > 7 THEN VAAN3% = 1
124 IF CHOICE% < 8 THEN VAAS3% = 0
125 IF CHOICE% > 7 THEN CHOICE% = CHOICE% - 8
126 REM * LETS PICKUP THIRD BIT *
127 IF CHOICE% > 3 THEN VAAN2% = 1
128 IF CHOICE% < 4 THEN VAAN2% = 0
129 IF CHOICE% > 3 THEN CHOICE% = CHOICE% - 4
130 REM * LETS GET SECOND BIT *
131 IF CHOICE% >1 THEN VAAN1% = 1
132 IF CHOICE% < 2 THEN VAAN1% = 0
133 IF CHOICE% > 1 THEN CHOICE% = CHOICE% - 2
134 REM * NOW PICKUP LAST SMALLEST BIT *
135 IF CHOICE% > 0 THEN VAAN0% = 1
136 IF CHOICE% < 1 THEN CHOICE% = 0
136 IF CHOICE% >0 THEN CHOICE% = CHOICE% - 1
137 REM * SO NOW CHOICE% IS 0 TILL NEXT INPUT *
138 REM * BITS ARE NOW SET TO 1 OR 0 *
139 REM * HERE A JUMP CAN BE DONE IF NEEDED *
```

So now after the Bits have been set we must get them transferred to the 74LS154 to get the info decoded and the line set to switch the relay and the connected device.

The correct order will be to PEEK the according soft-switches in the Apple-memory and thereafter execute a PEEK to the $C040 Softswitch that causes the 558 chip in the Apple II to perform a “strobe” signal at the gameport.

That “strobe”-signal starts the 74LS154 to ask for the signals at the inputs and decode them and then set the according output-line to “low”.

```
150 IF VAAN3% = 0 THEN POKE 49246,0
151 IF VAAN3% = 1 THEN POKE 49247,1
152 IF VAAN2% = 0 THEN POKE 49244,0
153 IF VAAN2% = 1 THEN POKE 49245,1
154 IF VAAN1% = 0 THEN POKE 49242,0
155 IF VAAN1% = 1 THEN POKE 49243,1
156 IF VAAN0% = 0 THEN POKE 49240,0
157 IF VAAN0% = 1 THEN POKE 49241,1
```

And now lets shoot out the “strobe”-signal

```
158 A = PEEK(49216)
```

The fact that you give the variable “A” a value from the adress 49216 performs a read-access to that adress and that causes the “strobe”-signal to be performed.

Now it would probably be nice to instruct the user, that the action has been executed:

```
160 HOME
```

161 PRINT " THE DEMANDED ITEM HAS BEEN SWITCHED ON."
162 PRINT " HIT ANY KEY TO CONTINUE"
163 PAUSE

Next we should determine if the program should end now or if we want this program to continue in a loop till the 0-key is pressed and that command ends up the program

Remember that in line 115 we set a branch to 2000 and that this line shall end the program. Unless no other functions are added to the program its now a good idea – if the program is to be continued – to clear the screen and bring up the choice-menu up to the display again.

The program will be extended with additional functions in a few days !

NOT complete yet !

This will be continued within the next days !

164 GOTO 1900
1900 HOME
1901 GOTO 80
2000 END

In the next Update step we will add to the program the additional feature to react on external incidents and act as alarm-system.

****

Up to this point the switching program is complete.

****
