



‘Mastermind on the KIM’

J. M. van der Peijl and D. M. de Boer

You are probably familiar with the well-known game Mastermind. Normally this game is played by two people, with only one person actively trying to guess a hidden code. The second person has the passive task of indicating, with black and white pegs, whether the chosen colours are correct and whether they are in the right positions. Replace the colours with digits, the black and white pegs with letters, and the KIM takes the place of the passive player. At your command, it chooses a secret colour combination and uses letters to indicate how many digits have been correctly guessed. A special “cheat button” makes it possible to peek secretly behind the screen; the consequence, however, is that the turn counter is increased by 30 points.

Instructions

Once you have entered the entire program (addresses 0200 ... 03D), you can start it at address 0200. The display will then go dark, and the KIM will store one digit (colour) in its memory each time you give the command. After a number of button presses, the secret combination is complete and the display lights up.

When choosing the combination, the following options are available:

Number of presses on the numbered key	Each digit is between	Maximum among the 4 digits
0	0..6	4 identical
1	0..6	3 identical
2	0..6	2 identical
3	0..6	4 different
4	1..6	4 different
5	0..6	4 identical
Other key	No action	

In this way, the game can be played at different levels of difficulty. When the display lights up, the KIM indicates that the combination is stored in memory.

The two rightmost displays form the turn counter; the four leftmost displays show the combination you have guessed. If you have pressed the wrong key, you can simply enter the combination again. Pressing E (enter) compares your guessed combination with the secret combination in memory. The KIM responds with a C for each correct colour that is not in the right position, and an A for each correct colour that is in the right position. The positions of the A's and C's do not indicate the positions of the correct colours. If you want to try the next combination, first press any key to increase the turn counter and clear the display.

In hopeless cases, pressing key C reveals the solution, while increasing the turn counter by 30. A correct solution displays "AAA", followed by the number of turns.

The display switches on and off a number of times and finally remains dark. A new secret code can now be made for the next game.

Scoring systems

After the editors had played a number of games on the KIM, it turned out that there are different methods for placing the white and black pegs.

Consider this situation:

0113 — secret code

3331 — guessed code

In this situation, the KIM responds with the letter C twice (two white pegs). The guessed code contains a 3 (in the wrong position), and the 1 is also a correct digit. There is, however, another system which says that the 3 has been guessed correctly three times, but the 1 once. This gives one correct digit in total, and therefore four white pegs. [The wording in the scan/OCR is unclear here.]

Are you used to this system? The KIM can adapt to your preference: set **code EA at address 0336**. (Normally address 036 contains code E4.) A third system says that the digit 1 occurs twice in the secret code and the digit 3 occurs once, making three white pegs in total. Set **code EA at address 032E** and the KIM will think as you do. (Normally E6 is found here.)

How it works

For those interested in the operation and construction of the program, a description of the Mastermind program follows. The program is divided into two important parts: the random-number generator (fig. 1), which makes the KIM choose a secret code without favouring particular combinations, and the comparison program (fig. 2). The latter part compares the guessed code with the secret code. We will discuss these two parts separately.

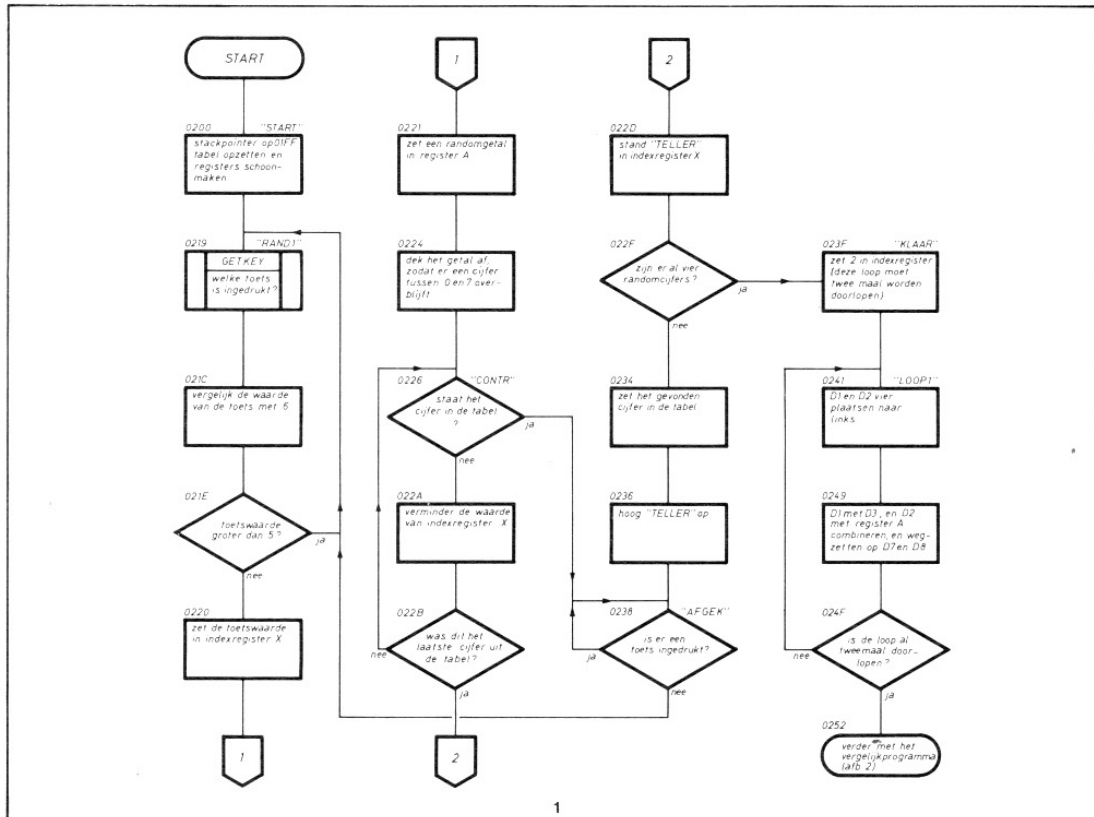


Figure 1, The random-number generator

RANDOM-NUMBER GENERATOR — PRINCIPLE

When a key is pressed, the program reads the KIM's timer (enter address 1706 to see it running). Since the press timing is independent of the timer, the value is treated as random. The resulting digit must, depending on the key, be between 0 and 6 or between 1 and 5. Since these values require only three bits, the first five bits of the random value are cleared, leaving 0–7 (00000000 ... 00000111). A table of forbidden digits is stored in memory, and each generated digit is checked against it. A match means rejection, requiring another key press:

Address	contents
00D5	06
00D4	00
00D3	FF
00D2	FF
00D1	FF
00D0	07

When choosing with key 0, only 7 (at 00D0) is excluded, leaving 0–6. With key 3, the candidate is compared in a loop with 00D3 ... 00D0. Suppose the first digit is 3: it is compared with FF, FF, FF and 07, accepted, and stored at 00D1. It is added to the forbidden table, so 3 cannot recur. If the next press produces 1, it is compared with FF, 03 and 07, accepted, and stored at 00D2. If 1 is produced again, it is rejected because the table now contains 01, 03 and 07. The third accepted digit is stored at 00D3. The fourth is checked but need not be added to the table. Choosing with key 5 also excludes 0 and 6. At least four presses of keys 0–5 are needed; rejected values require extra presses. Rejections are not shown, but completion is. The processor then enters the comparison program; the display lights with the turn counter at 1.

RANDOM-PROGRAM FLOWCHART

At 0200 the counters and table 00D0–00D5 are initialized. At 0219 the pressed key is determined. If none is pressed, A is set to \$15 (\$ denotes hexadecimal), and the loop repeats until a key 0–5 is pressed. The key value is put in X at 0220. The timer value is placed in A and masked with 00000111, leaving 0–7. The checking loop starts at 0226. Depending on X, the candidate is compared with the appropriate part of the table. A match branches to AFGEK at 0238, waits for release, and returns to the program start.

If no match is found, the counter at 00D6 is checked. If fewer than four digits have been accepted, the digit is stored, the counter incremented, and after key release the program returns to “rand 1”. After four digits, the program combines them at 0241. The first three digits are at 00D1–00D3 and the fourth is in A. For example, if these locations contain 01, 02, 03 and A contains 04, X is set to 02 at 023F. The value at 00D2 is shifted four places left, turning 02 into 20. At 0249 it is added to A: $4 + 20 = 24$, stored at 00D8. A is then loaded from 00D3 (03), X is decremented to 01, and the loop repeats. The contents of 00D1 are shifted and added, producing 13, stored at 00D7. X becomes 00, so the loop ends and comparison begins.

COMPARISON PROGRAM — PRINCIPLE

Only digits 0–6 can be entered. Each new digit shifts the digits already on the display one place left. The 16-bit display value (four hexadecimal digits) is shifted by four bits using SHIFT. The input section can be left only with C or E. Pressing C copies the secret code to POINTO and POINTH so it can be displayed, loads 29 into A, and calls the increment routine. This sets decimal mode and carry, increasing the turn counter by 30. Input then resumes. Pressing E leaves input and starts comparison, which displays A’s and C’s using VERG and SHIFT.

Example:

4341 secret code

4313 guessed code

Put 0A at 00E2 and call VERG. It finds two matches in the first and second digits (4 and 3), shifting two A's into the display. To prevent recounting, matching secret-code digits are increased by 8 (making values 8–E), and matching guessed-code digits are replaced by F:

CB41 secret code
FF13 guessed code

Next, diagonal matches are checked. Put 0C at 00E2 and rotate the guessed code one place:

CB41
F13F

No match. Rotate again:

CB41
13FF

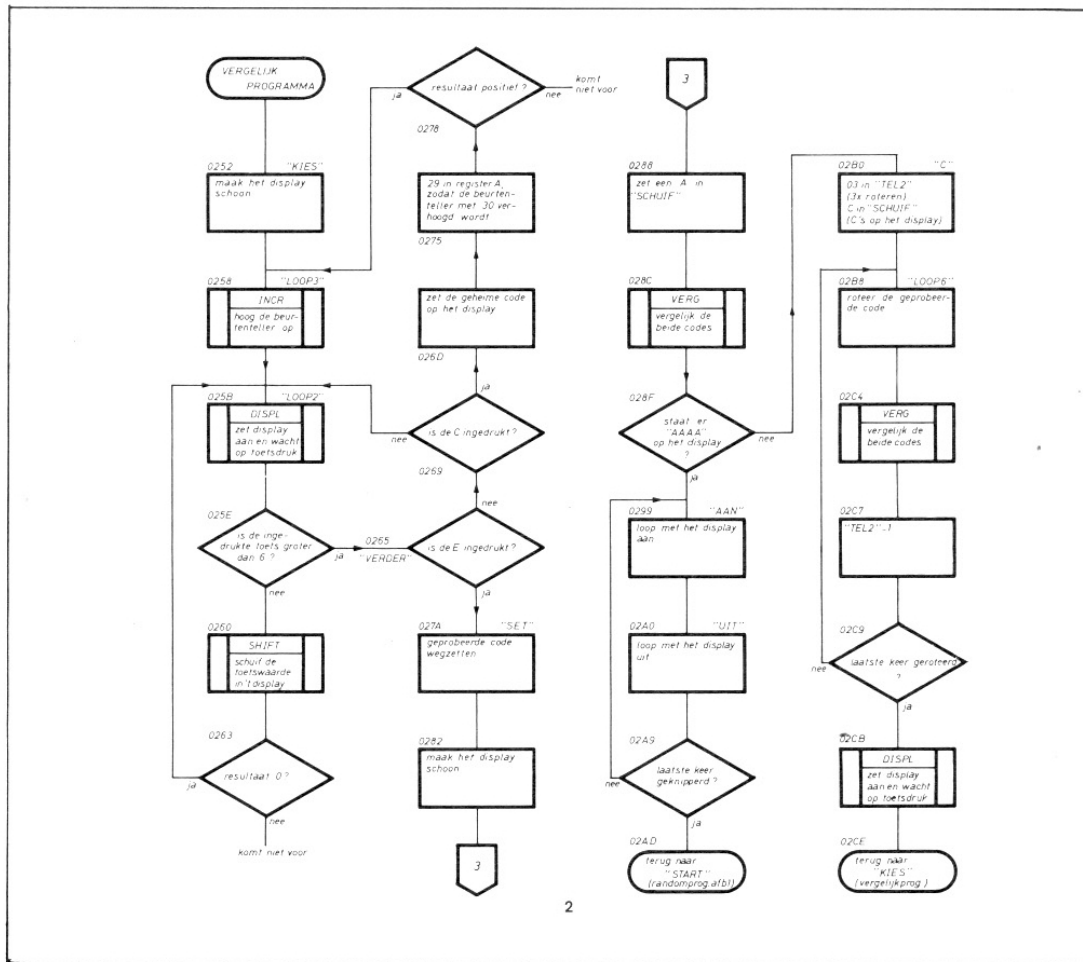
Again no match. Without masking during the first comparison, the codes would have been:

4341
1343

This would falsely produce two C's. The final rotation is:

CB41
3FF1

The fourth digit (1) matches, so C is shifted into the display. Another rotation would return to the starting position. The result is "0AAC". The display subroutine makes this stored result visible. After a key press, comparison restarts, the display is cleared and the turn counter incremented. Four A's mean the game is over and the main program restarts for a new secret code.

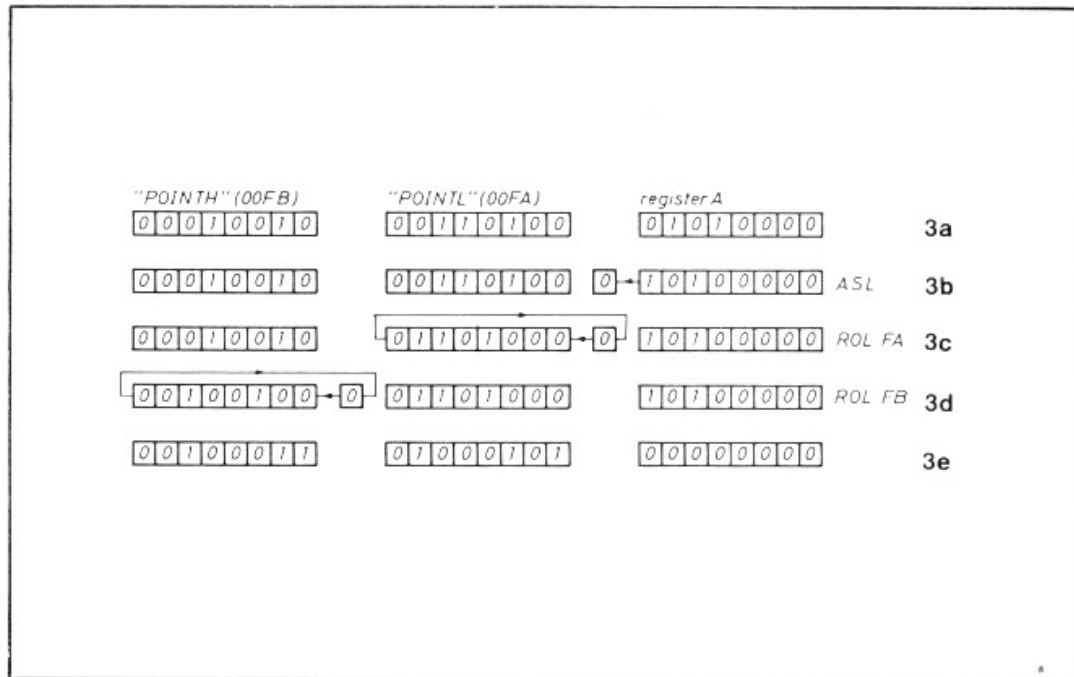


COMPARISON-PROGRAM FLOWCHART

At 025B the processor calls DISPL and waits for a key. Keys 0–6 go via 025E and 0260 to SHIFT, which shifts the display left and puts the new digit at the right. On return, the Z bit is set; BEQ at 0263 returns to loop 2 (025B). Keys greater than 6, except E and C, return via 0265 and 0269 without action. E leaves input and begins comparison at 027A. The guessed code is copied from display memory because POINTLO and POINTHI will hold the A's and C's. At 0282 display memory is cleared to zero. An A is placed in SHIFT at 00E2, and VERG is called at 028C to mark correctly positioned digits. Matches are altered to prevent duplicate detection.

If display memory does not contain "AAA", the program jumps to C at 02B0 and puts 03 at 00E1. This counter records the number of rotations. Rotation starts at 02B8; the codes are compared at 02C4 and the counter decremented. The process repeats three times, adding a C to the display for each match. Then the program returns to DISPL. A key press restarts the comparison section at 0252: display memory is set to 0000, the turn counter is incremented, and the original secret code is restored at 00E5 and 00E6 because the previous comparison altered it. Input resumes at 025B. If the code is

correct, four A's are detected at 028F. The display is lit in a loop at 0299 and turned off in a loop at 02A0. This is repeated six times. Location 00D4 is a counter that counts down from 00 through FF to 00; 00D5 counts the flashes. After the final flash, the program restarts and a new secret code can be created.



DISPLAY-SHIFT DIAGRAMS (FIG. 3)

3a — Register state after A has been shifted left four times; display memory contains 1234.

3b — ASL: bit 7 shifts into Carry.

3c — ROL FA: the Carry bit shifts into bit 0 of POINTL; bit 7 of POINTL returns to Carry.

3d — ROL FB: bit 7 of POINTL enters bit 0 of POINTH.

3e — After four rotations, display memory contains 2345.

THE SUBROUTINES

The four subroutines are INCR (increments the turn counter), DISPL (activates the display and identifies the key), SHIFT (shifts a digit in A into the display), and VERG (compares the codes).

INCR (address 02D1)

INC does not operate in decimal mode, so ADC is used to increment the turn counter. Set the decimal-mode flag first, and set Carry as well. The value added is A + the old counter value + 1. To increment by one, A must contain 00. The routine also copies the secret code to 00E5 and 00E6. It is called only once, so a subroutine is not strictly necessary, though extensions such as penalty points could use it.

DISPL (address 02E2)

This routine continually displays the digits in POINTLO and POINTHI and determines whether, and which, key is pressed. It uses the monitor routine SCANDS. On return from SCANDS, A is 00 if no key is pressed. The first loop (02E2–02E6) waits until no key is held, since the previous key may still be down. The second loop (02E7–02EB) waits for a press. Addresses 02EC–02F0 check again that a key was really pressed, since a disturbance could also exit the loop. Address 02F1 identifies the key. Comparing it with \$07 sets the N bit for keys 0–6, which can then be used for conditional branches.

SHIFT (address 02F7)

Suppose the display shows 1234 and A contains 05. At 02F7–02FA, A is shifted left four times, producing 50. At 02FD A is shifted once more; the bit shifted out enters Carry. The first ROL (02FE) shifts that bit into POINTLO, while POINTLO's outgoing bit enters Carry. ROL at 0300 shifts that bit into POINTHI. The value has moved one place left. Repeating this in a loop shifts all bits four places, producing display value 2345.

VERG (address 0306)

VERG compares two codes. It has three sections: frame (0306–0314), comparison (0315–0328), and masking (0329–033D). Each memory location contains two digits, so one digit must be masked at a time. The frame first stores \$F0 at 00E0. The comparison section uses AND with this value to zero the right-hand digit and compare the left-hand digit. At 030A it jumps to 0315; since this section runs twice, X is set to 02. A part of the secret code (from 00E6) is loaded into A and masked with F0, leaving one digit, temporarily stored at 00E0. At 031D the corresponding part of the guessed code (00E8) is loaded and masked; the digits are compared at 0321. The section runs again to compare the left digits from 00E5 and 00E7. The program returns to the frame at 030D, stores \$0F rather than \$F0 at 00E0, and repeats the process for the right-hand digit. A match branches to the masking section at 0329. The relevant secret-code digit is masked. At 032F and 0331, A is set to \$80 or \$08. ORA adds this bit to the corresponding guessed-code digit. Finally, the value at 00E2 (A or C) is shifted into the display with SHIFT.

EXTENSIONS

Try changing the program—for example, use E and F instead of A and C as the code pegs. You could make the turn counter stop at 99, preventing cheating with the C key (pressing it three times makes the counter show 10 fewer points).

For advanced users: make the turn counter increase automatically after every 30 seconds of thinking time, adding tension to the game. You could also make the program keep scores for different players automatically. Programming is a hobby in which instructions cost nothing, and incorrect instructions cannot blow up transistors.