



D.M. de Boer

De Melodiant



The Melodiant plays a complete melody of up to 256 tones on a command given by you. The display shows which note is being sounded. This makes the Melodiant a very versatile instrument. For example, you can play duets together with the Melodiant by typing in one of the two voices beforehand. The advantage over a tape recorder is that the melody can be played at any desired tempo, which can make practicing easier. What about the Melodiant as a music teacher? A note appears on the display, and the student must try to sing or whistle that note. Afterwards, the loudspeaker can be switched on to check. The Melodiant will also flawlessly play back the most difficult rhythms, slow or fast as needed.

But the possibilities do not end there. We can make the Melodiant play so fast that the individual tones are no longer recognizable. This can produce the strangest sound effects.

The flowchart and the program

When discussing the Melodiant, we always make use of a flowchart and a program. The flowchart clearly shows all the actions the microprocessor must perform.

In the program, on the other hand, we can clearly see which instruction is at which address. In addition, some memory locations have been given a name, because a name is easier to remember than an address. In the program (see list 1) we find six columns. The first column gives the address. The second column gives the contents at that

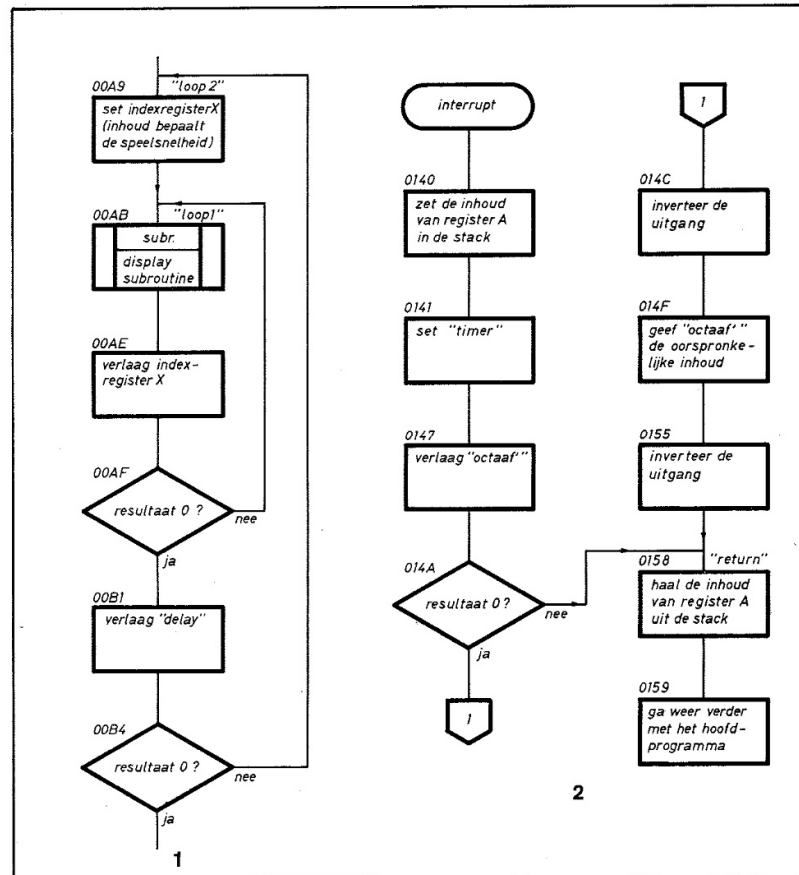
memory location. For programming purposes, these two columns are in principle sufficient. However, if we want to follow and understand the program, we also need the other columns. The third column contains the name a memory location may have; this name is also called a "label" (most addresses have no name). The fourth column gives, in abbreviated form, the instruction to be executed by the microprocessor, while behind it, in the fifth column (in italics), is the addressing technique used. An explanation of the abbreviations and addressing techniques can be found in the general discussion of the KIM-1. Below the italicized text is always the name (or address) of a memory location where the operand is stored. Only in the case of immediate addressing is the operand itself given here. The \$ sign indicates that the number is written in hexadecimal. The last column provides commentary on the various program steps. This column also links back to the flowchart, since roughly the same terms are used in the flowchart. As mentioned, the flowchart gives an overview of the order in which the various parts of the program are executed (fig. 4). Above each block is an address, along with, where applicable, a name for that address. This address indicates which piece of program the block in question relates to. In this way, any part of the program can always be easily found.

Operation

The KIM-1 is programmed so that it generates a pulse signal of the correct frequency on the connection PA5, which is set as an output. For each tone to be produced, the program must set the frequency on this output to the correct value, give the tone the correct duration, and finally place the name of the note on the display. Now, the microprocessor can do an enormous amount, but it cannot do more than one thing at a time. We therefore proceed as follows: first, a small piece of program ensures that one of the six displays shows the correct symbol. Each time this piece of program is executed, the next display lights up (the display is multiplexed). The time the microprocessor needs to run through this piece of program (just over 1 ms) is now used as the time unit for the duration of the tone to be played. To achieve a reasonable duration, this display subroutine is executed a number of times. This is achieved by placing a certain number in the index register (fig. 1). After running through the display routine, the index register is then decremented by one. As long as the result is not '0', the program keeps jumping back to the display subroutine. In this way, depending on the initial content of index register X, the display subroutine can be executed 1 to 256 times, achieving a duration of roughly 1 to 256 ms. Because this duration is still much too short, the whole process is repeated once more. This time, however, not using index register X, but a location in external memory. We call this memory location "delay" (address 0111). We can now again run through the first loop 1 to 256 times, so that with these two loops the note duration can be set from about 1 ms up to just over 1 minute. The memory location "delay" is continually filled by the program with the desired note duration. By changing the initial content of index register X (at address 00AA), the

playback speed can be adjusted.

The piece of program associated with these loops is located at addresses 00A0 through 00B8. For clarity, these addresses are also shown in the flowchart in fig. 1.



1. The flowchart of waitcycle 2. This takes care of the length of the note played
2. The flowchart of the interrupt program . This program runs completely independent of the main program.

Producing a tone

We have now seen how the program ensures the tone gets the correct duration. That, however, is not enough. We also need to produce a tone of the correct frequency. The principle is very simple. For each note, the oscillation period $T (=1/f)$ has been calculated. Each time this period has elapsed, the output of the Melodiant is briefly set to '1'. In this way, a pulse voltage of the correct frequency is produced at the output. There is, however, a problem. During the time the tone must sound, the program is busy displaying the name of the note and determining the correct duration of the note. For this reason, the generation of the tone is "outsourced." The KIM-1 has an interval timer. This timer can be loaded by the program with a particular value. From that moment on, the timer operates completely independently of the program. After the programmed

time has elapsed (in this case, the oscillation period T of the note in question), an "interrupt request" is generated. This means that the running program is briefly interrupted and a second program is automatically started.

This second program has a run time of about 50 μs . The flowchart is shown in fig. 2, while the interrupt program is given in list 2. In fig. 2, too, the program addresses are noted, so the reader can easily follow the program. The interrupt program begins by storing the contents of register A. After all, when the main program is resumed, all registers must be restored to their original state. Next, the timer is restarted, so that another interrupt can occur one oscillation period T later.

What has not yet been discussed is how tones in lower octaves are generated. We always start from the oscillation periods of the highest octave. Lower octaves are then produced by dividing that highest tone by 2, 4, or 8. How many times the frequency must be divided is stored in the memory location "octave" (address 0114). Depending on the content of this memory location, a pulse is placed on the output every time the interrupt program runs ("octave" = 1), or this happens only once every 2, 4, or 8 times ("octave" = 2, 4, or 8).

Going back to fig. 2, we see that "octave" is decremented by 1 (address 0147). Only if the result is '0' is the output inverted, "octave" is refilled with its original content, and the output is inverted again. In this way, a pulse of 12 μs is produced at the output once every 1, 2, 4, or 8 periods. Once this has happened, the content of register A is restored, so that the main program can simply continue. The final instruction, "return from interrupt," ensures that the main program continues as if nothing had happened.

How the music is stored in memory

We have now seen how our computer determines the duration and frequency of the tone. However, the program does not do this on its own. It always looks in a special reserved section of memory to determine which note should be played and for how long. This section of memory runs from address 0200 through 03FF.

The total space for the melody is thus 512 bytes. Each tone takes up 2 bytes, so that 256 tones (or rests) can be programmed. The code is chosen as follows: the first byte encodes the pitch, the second byte the duration. For the pitch, the code is based on the "ordinary" C scale:

- Rest 0 — stop 8
- C 1 — C sharp 9
- D 2 — D sharp A
- E 3 — E sharp B
- F 4 — F sharp C
- G 5 — G sharp D
- A 6 — A sharp E
- B 7 — B sharp F

For sharps, 8 is added (in hexadecimal) to the code of the note in question. This has several advantages. First of all, it is easy when typing in the music, since the sharps are always 2 keys higher than the original note. Suppose we want to enter an F sharp. People with a bit of musical knowledge know that F is the 4th note of the scale; F sharp is then 2 keys higher. At first this takes some getting used to, but coding a melody becomes faster and faster over time.

So far we have only used the units digit of the first byte to encode the note in the highest octave. We already mentioned that lower octaves are obtained through division. How many times the division must occur can now be indicated in the tens (hexadecimal "16s") digit of the first byte (hexadecimal because we work in the base-16 system). The octaves are thus indicated with a 1 for the highest octave, and 2, 4, or 8 for lower octaves. An E in the lowest octave thus becomes: 83; an F in the second-highest octave becomes 24; a G in the highest octave becomes 15.

Two more important codes are used: 0 gives a rest, where the length of the rest is determined in the same way as for a note. The code 8 is a stop marker and must be placed at the end of the melody.

The duration of the note is stored in the second byte. Here we can fill in values from 01 to FF. Coding the duration is very simple. Suppose the fastest note is a 1/32 note (3 flags on the stem). This note is given the value 01. A 1/16 note then becomes 02, an 1/8 note becomes 04, a 1/4 note becomes 08, a 1/2 note becomes 10 (in hexadecimal, 08 is half of 10!), and a whole note becomes 20. Even complex rhythms with triplets can be programmed by dividing each bar into a sufficient number of small parts.

We would like to emphasize once more that this memory area (0200–03FF) is not part of the program. This section of memory should be regarded as the "sheet music" from which the computer plays.

Decoding the music

Before a note can be sounded, the codes must be translated by the computer. The various quantities of a given note are each placed in a set of fixed memory locations, namely:

Name	address
T	0110
Delay	0111
Octave	0112
octave'	0114
display	011C
pitch	0120
tone	0122
I	0123
S	0124

In "T," the oscillation period is finally stored; in "delay," the duration of the note; in "octave" and "octave," the number of divisions. Here, "octave" is continually decremented by 1 during the program down to 0, while "octave" retains its value. During playback of a melody, the value C is placed in "display." This number designates a series of memory locations in which the quantities to be displayed are already stored in coded form. Two of these memory locations are continually given a new value during the music: these are the memory locations "pitch" and "tone," which contain, respectively, the display code for the octave and the display code for the tone. The flowchart in fig. 4 shows the entire main program. The last part of this (from address 00A9) is the actual music program and has already been discussed under "Operation." The decoding section begins at address 0042. Using the flowchart (fig. 4), we will go through the various operations.

The first thing the program does is set the interrupt flag. This means that interrupts are ignored from this point on. No more pulses will appear on the output either (the pulses were generated by the interrupt program). The previous tone is thus effectively switched off. Next, the first byte of the new tone is placed in register A. Index register Y keeps track of which tone is up next. Indirect indexed addressing is used here, meaning that the second byte of the instruction (address 0044) gives an address on page 00 (00EA). The contents of this address are added to the contents of index register Y. The result of this addition is the final address on a page which, in this case, is stored at address 00EB. In our case, this works out as follows: index register Y gives the address on page 02 (the content of 00EA is 00, the content of 00EB is 02). After each cycle, index register Y is incremented, so that the correct tone is always fetched.

Once the pitch code is in register A, an "AND" operation is performed with the binary number 00001111. This means that after this operation the tens digit becomes 0 and only the units digit remains. The code indicating the octave has thus disappeared (a 0 in an input always produces a 0 at the output). Next, the remaining code is compared with 08. If the remaining number is indeed 8 (stop code), the Z-bit in the status register will become '1' and the program will jump to the "ready" cycle. If it was not the stop code, we simply continue and the note code is placed in memory location T (address 0110). This code is later translated into an oscillation period.

Next, another AND operation is performed, this time with 00000111 (binary). As a result, the information indicating whether the note was sharpened or not is also discarded. The remaining code can now range from 0 to 7 and stands for the letters C through B. This code is used to place the corresponding letter on the display. The display routine looks at memory location "tone" to display this letter. To get a particular letter onto the display, we must place the correct code in this memory location. The code for "C," for example, is 39. So we need to convert the codes 0 through 7 into codes the display can interpret. For this purpose we have created a table in computer memory containing these codes. The table runs from address 012F through address 0136 (see list 3). How do we get the correct code? Well, for this we use indexed addressing. First, we

place the code that is in register A into index register X (address 0051, see the program). At address 0052, the content of a memory location is then placed into register A. The address of this memory location is the sum of the content of index register X and 012F. The desired code is now at the addressed memory location. This code is first placed into register A (address 0052), and then into its final destination "tone" (address 0122). As an example, let's take the note D sharp in the lowest octave. The code in memory is: 8A. At address 0046 (fig. 4 and list 1), the "AND" operation is performed. After this operation, the remaining code is 0A; this code is placed in "T" (address 004C). Now the AND operation is performed a second time, and again a bit is set to '0'. Of 0A, only 02 now remains. This 02 is the code for "D." At address 0051, this 02 is placed into index register X. At address 0052, the content of address $02 + 012F = 0131$ is placed via register A into memory location "tone," with the result that a "D" appears on the display. The suffix "sharp" is handled by the piece of program that follows.

First, "00" is placed in memory locations "i" and "s." These memory locations are used by the display routine for the 4th and 5th displays. On these displays, the letters "is" (Dutch for "sharp") must appear for sharpened notes. By placing a '0' in these memory locations, the display is switched off. So initially we assume we are dealing with a natural (non-sharpened) note. At address 0061, the code (0A in our example) that was in "T" is placed back into register A. The AND operation is performed again, this time leaving all bits at '0' except one. The one remaining bit is '1' for a sharpened tone and '0' for a non-sharpened tone. In fig. 4 (address 006A) we see that if the tone is sharpened, the code for "sharp" is placed at addresses 0123 and 0124, for the letters "i" and "s" respectively. If there was no sharp, this section is skipped, so that the corresponding displays remain dark. The necessary codes for displaying the note have now been taken care of.

Next, we decode the octave code. First, the complete code is again fetched from memory (address 0076). Through an AND operation, the units digit is now removed (this AND operation is also called "masking"). From a code of 8A, 80 would now remain. Next, we shift 4 times to the right, so that the tens digit ends up in the units position (80 thus becomes 08). The resulting code is stored in "octave" and "octave'." Next, once again a table is consulted to determine which display code should be used. This table is located in memory at addresses 0137 through 018F. The display code is stored at address 0120 ("pitch"), and the display routine ensures that an indication of the octave appears on the first display.

We are now at address 008D. Here, index register Y is incremented, so we now access the second byte of the programmed note. This code is also fetched and placed in memory location "delay." This code does not need to be further decoded. We still need to convert the code now in "T" (a number between 00 and 0F) into an oscillation period. This, too, is done using a table. The table is located at addresses 0100 through 010F (list 3), and the decoding is done in the now-familiar way. The numbers in the table are hexadecimal and give an oscillation period as a multiple of 8 μ s. The KIM-1's timer is set

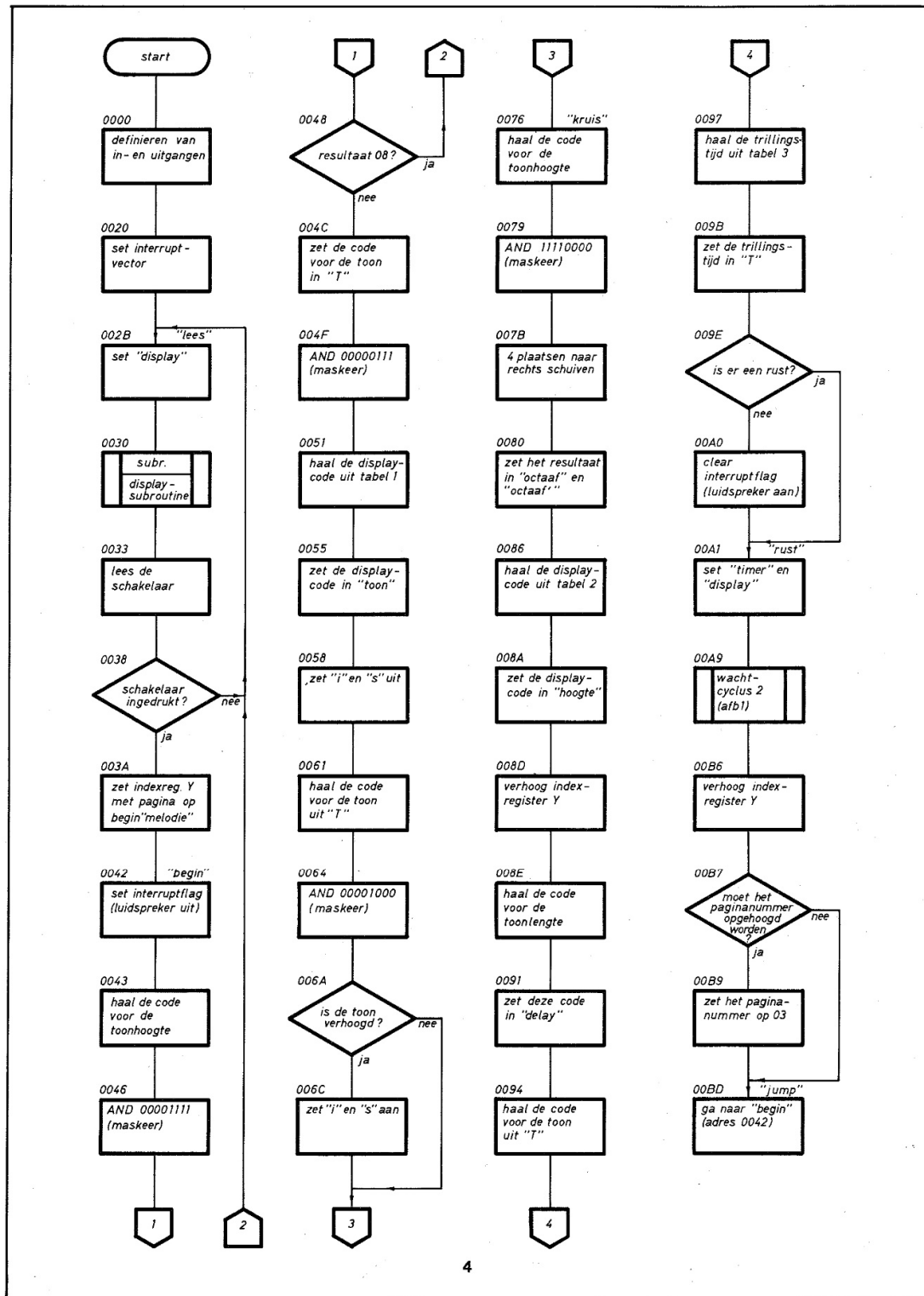
by the program to a count period of 8 μ s.

At address 009E, finally, a check is made to see whether we are dealing with a rest. In that case, "T" contains the value '00'. If there is no rest, the interrupt flag is set to '0', so that interrupts can occur again. As a result, the new tone will sound through the loudspeaker. If there is a rest, the interrupt flag remains '1', so that it stays silent. Because for the very first tone an interrupt can only occur after the timer has been set, the timer is started once at address 00A1. On subsequent occasions, this is taken care of by the interrupt program.

As the final step of this decoding section, memory location "display" is filled with the value 1C, so that the display routine makes the correct section of memory visible. From this point on, the actual program begins, which was already discussed under "Operation."

The beginning of the program

Almost all programs must begin by defining various things. Since this is the least interesting part (though no less important!), we will treat it briefly. First, the various connections of the KIM-1 are defined as input or output. For this, a '1' is written into a special data direction register if the corresponding connection is to serve as an output, and a '0' if it is to be an input. At address 20, the interrupt vector is defined. In other words: here the microprocessor is told that upon an interrupt it must jump to address 0140, where the interrupt program begins.



Wait cycle 1

Wait cycle 1 begins at address 002B. First, the value '12' is placed in memory location "display" (address 011C). As a result, the display routine will not show a played note, but instead the word "READY," which is stored in coded form in memory locations 0116

through 011B (list 3). Next, the program jumps to the display subroutine, which ultimately makes the word "READY" appear on the display. At address 0033, all connections PA0 through PA7 are read. The intention is that we only read the state of the start switch. Therefore, all other bits are forced back to '0' (through the AND operation). As long as the switch is not pressed, this piece of program keeps repeating. Upon a start signal, after index register Y is set to '0' and the page to 02 (the melody begins at address 0200), the decoding section begins.

The display subroutine

Each time this subroutine is called, only 1 of the 6 displays is turned on. Because a different display is taken each time the subroutine is called, we see 6 displays lighting up (in reality, the displays are multiplexed). To fully understand the program of this subroutine, it would first be necessary to explain in detail how the displays are connected to the KIM-1. Since this is beyond the scope of this article, we will only briefly discuss the flowchart. In a later article we will go deeper into the displays and the keyboard.

The flowchart for this subroutine is shown in fig. 5. We begin by storing the contents of register A and index register Y, so that these registers can be freely used by the subroutine. Next, a counter is incremented, which ensures that the next display is now activated. If this counter reaches 7 (there is no display 7), the counter is reset to 1. Next, it is determined at which memory location the display code is stored. This is determined based on the content of memory location "display" and the counter mentioned earlier. After all, each display must show a different symbol. This display code is then sent to the display, so that the desired symbol appears. After this, a wait cycle begins, producing the desired delay of about 1 ms. This delay is created using a loop; the principle has already been discussed under "Operation." Finally, registers A and Y are restored to their original state, and the program jumps back to the main program.

The complete program

We have now discussed the entire program piece by piece. We deliberately did not start at the beginning, but at the core of the program. This has the advantage that the program can be presented in a logical structure. For an overview, we will list the parts once more in the correct order.

The main program consists of:

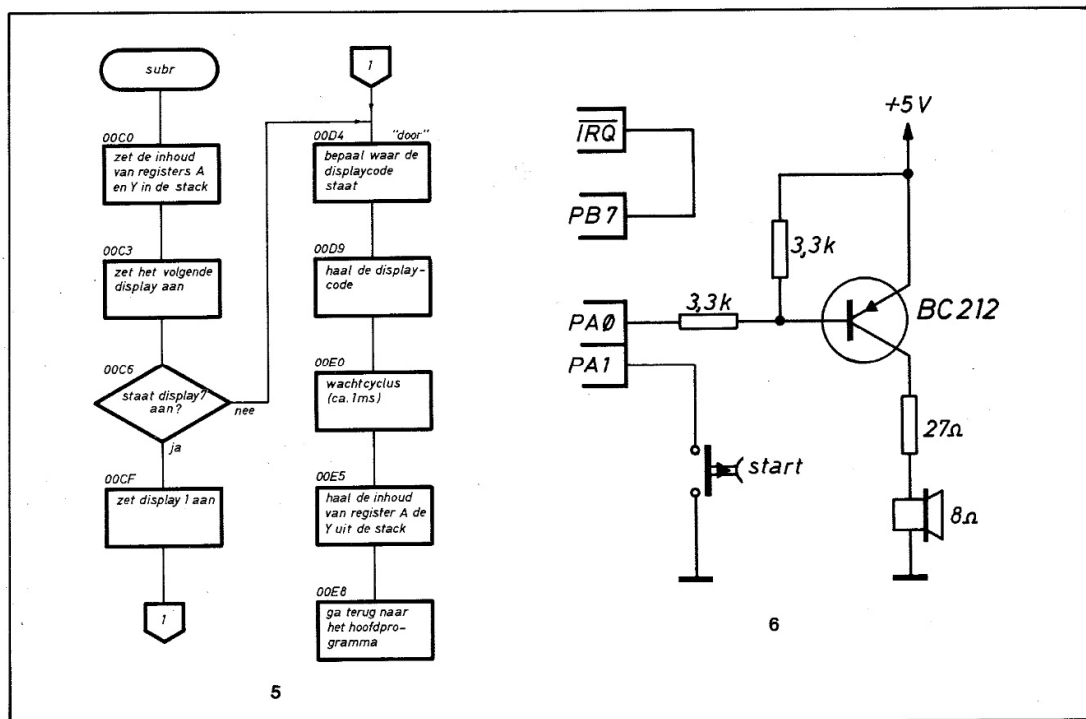
- a. The beginning, at addresses 0000 to 002A. This part defines the inputs and outputs and the interrupt vector (fig. 4 and list 1).
- b. Wait cycle 1, at addresses 002B to 0039. This part runs as long as the start button has not been pressed. During this cycle, the program repeatedly jumps to the display subroutine so that the word "READY" appears on the display (fig. 4 and list 1).
- c. The decoding section, at addresses 0042 to 00A8. At the start of each tone, this section is run through to fill various memory locations with data for the tone (duration,

oscillation period, display codes) (fig. 4 and list 1).

d. Wait cycle 2, at addresses 00A9 to 00B8. This part takes care of the duration of the note and regularly calls the display subroutine, which in turn places the name of the note on the display (fig. 1 and list 1).

e. The display subroutine, at addresses 00C0 to 00E8. This subroutine makes the display codes stored in memory visible on the displays (fig. 5 and list 1).

f. The interrupt program (fig. 2 and list 2). This is a completely independent program, started by the timer. This program generates the pulses at the output and, in turn, restarts the timer.



The external components

Although it goes against the grain for a true programmer, we will actually have to pick up the soldering iron. We need to mount a small loudspeaker and a push-button on the connections of the KIM-1. We also need to connect the output of the timer to the IRQ (interrupt request) input. Fig. 6 shows how everything should be connected. For the loudspeaker, a small transistor amplifier has also been included, but of course you can also connect the Melodiant to your own (hi-fi) amplifier.

Instructions for use

After you have connected all the components to the KIM-1, and have typed in the numbers from lists 1 through 3 (the numbers from the second column must be placed at the addresses given in the first column), the program can be started. We enter address 0000, then press RS (reset) and GO. If everything is correct, the word "READY" will now appear on the display. If we now press the external button, the melody (if you

have already entered one) will sound from the small loudspeaker.

If you want it slower or faster, press ST (stop) and enter address 00AA. The content of this address must be made larger if you want it slower, and smaller if you want it faster. Restarting is done in the same way, and the melody will be played by the Melodiant at whatever tempo you like.

What will this cost?

Finally, a question important to many readers: "What will this cost?" If you already have a KIM-1, the costs are very low. If you don't have a junk box and need to buy the parts new, reckon on about f 10 [10 guilders]. If you don't yet have a KIM-1, it wouldn't be fair to simply add fl 998 [998 guilders] (for subscribers) to that tenner — after all, your KIM-1 is not just a Melodiant! Besides being a Melodiant, the KIM-1 can also be a clock, darkroom timer, chess computer, calculator, and so on.

Lijst 1 Het hoofdprogramma.

Hoofdprogramma							
0000	A9	LDA	immediate	002A	EA	NOP	implied
0001	01		\$01	002B	A9	lees	LDA immediate
0002	8D	STA	absolute	002C	12		\$12
0003	01		PADD1	002D	8D	STA	absolute
0004	17		1701	002E	1C		display
0005	A9	LDA	immediate	002F	01		011C
0006	7F		\$7F	0030	20	JSR	absolute
0007	8D	STA	absolute	0031	C0		subr.
0008	41		PADD2	0032	00		00C0
0009	17		1741	0033	AD	LDA	absolute
000A	A9	LDA	immediate	0034	00		PAD1
000B	00		\$00	0035	17		1700
000C	8D	STA	absolute	0036	29	AND	immediate
000D	40		PAD2	0037	02		\$02
000E	17		1740	0038	D0	BNE	relative
000F	A9	LDA	immediate	0039	F1		lees
0010	1E		\$1E	003A	A0	LDY	immediate
0011	8D	STA	absolute	003B	00		\$00
0012	43		PBDD2	003C	EA	NOP	implied
0013	17		1743	003D	EA	NOP	implied
0014	A9	LDA	immediate	003E	A9	LDA	immediate
0015	08		\$08	003F	02		\$02
0016	8D	STA	absolute	0040	85	STA	zero page
0017	42		PBD2	0041	EB		00EB
0018	17		1742	0042	78	begin	SEI implied
0019	78	SEI	implied	0043	B1	LDA	(ind), Y
001A	EA	NOP	implied	0044	EA		melody
001B	EA	NOP	implied	0045	EA	NOP	implied
001C	EA	NOP	implied	0046	29	AND	immediate
001D	EA	NOP	implied	0047	0F		\$0F
001E	EA	NOP	implied	0048	C9	CMP	immediate
001F	EA	NOP	implied	0049	08		\$08
0020	A9	LDA	immediate	004A	F0	BEQ	relative
0021	40		\$40	004B	DF		lees
0022	8D	STA	absolute	004C	8D	STA	absolute
0023	FE		17FE	004D	10		T
0024	17			004E	01		0110
0025	A9	LDA	immediate	004F	29	AND	immediate
0026	01		\$01	0050	07		\$07
0027	8D	STA	absolute	0051	AA	TAX	implied
0028	FF		17FF	0052	BD	LDA	abs, X
0029	17			0053	2F		tabel 1
				0054	01		012F

0055 8D	STA	absolute	display code in 'toon'	00A0 58	CLI	implied	luidspreker aan
0056 22		toon		00A1 8D rust	STA	absolute	
0057 01		0122		00A2 0D		timer	set timer (8 µs)
0058 A9	LDA	immediate		00A3 17		170D	
0059 00		S00		00A4 A9	LDA	immediate	
005A 8D	STA	absolute	zet 'i' en 's' uit	00A5 1C		\$1C	set 'display'
005B 23		i		00A6 8D	STA	absolute	
005C 01		0123		00A7 1C		display	
005D 8D	STA	absolute		00A8 01		011C	
005E 24		s		00A9 A2 loop 2	LDX	immediate	
005F 01		0124		00AA 20 snelh.		\$20	
				00AB 20 loop 1	JSR	absolute	
0060 EA	NOP	implied	code voor de toon in A	00AC C0		subr.	
0061 AD	LDA	absolute		00AD 00		00C0	
0062 10		T		00AE CA	DEX	implied	begin wachtcyclus 2 2 loops
0063 01		0110	00AF D0	BNE	relative		
0064 29	AND	immediate	maskeer	00B0 FA		loop 1	
0065 08		\$08		00B1 CE	DEC	absolute	
0066 EA	NOP	implied		00B2 11		delay	
0067 EA	NOP	implied		00B3 01		0111	
0068 EA	NOP	implied		00B4 D0	BNE	relative	
0069 EA	NOP	implied		00B5 F3		loop 2	
006A F0	BEQ	relative	is er een kruis?	00B6 C8	INY	implied	indexreg. Y ophogen moet het paginanr. opgehoogd worden?
006B 0A		kruis		00B7 D0	BNE	relative	
006C A9	LDA	immediate		00B8 04		jump	
006D 06		\$06		00B9 A9	LDA	immediate	verhoog het paginanummer van 02 naar 03 terug naar het begin
006E 8D	STA	absolute	00BA 03		\$03		
006F 23		i	00BB 85	STA	zero page		
				00BC EB		00EB	
0070 01		0123	zo ja,	00BD 4C jump	JMP	absolute	
0071 A9	LDA	immediate	zet 'i' en 's' aan	00BE 42		begin	
0072 6D		\$6D		00BF 00		0042	
0073 8D	STA	absolute		00C0 48 subr.	PHA	implied	begin display subroutine
0074 24		s		00C1 98	TYA	implied	
0075 01		0124		00C2 48	PHA	implied	
0076 B1 kruis	LDA	(ind), Y	haal de	00C3 EE	INC	absolute	hoog de teller op voor het volgende displ.
0077 EA		melody	toonhoogte	00C4 42		PBD2	
0078 EA	NOP	implied		00C5 17		1742	
0079 29	AND	immediate	maskeer	00C6 AD	LDA	absolute	
007A F0		\$F0		00C7 42		PBD2	
007B 4A	LSR	accum.		00C8 17		1742	
007C 4A	LSR	accum.	schuif de 16-tallen naar de eenheden	00C9 29	AND	immediate	display ??
007D 4A	LSR	accum.		00CA 1E		\$1E	
007E 4A	LSR	accum.		00CB C9	CMP	immediate	
007F EA	NOP	implied		00CC 14		\$14	
				00CD D0	BNE	relative	zo ja, naar 'door'
0080 8D	STA	absolute		00CE 05		door	
0081 12		0112	set octaaf	00CF A9	LDA	immediate	
0082 01		0114	en octaaf				
0083 8D	STA	absolute		00D0 08		\$08	zo nee, zet teller op display 1
0084 14		0114		00D1 8D	STA	absolute	
0085 01		0137	haal de	00D2 42		PBD2	
0086 AA	TAX	implied	displaycode	00D3 17		1742	
0087 BD	LDA	abs, X	uit de tabel	00D4 4A door	LSR	accum.	
0088 37		tabel 2		00D5 18	CLC	implied	bepaal de geheugenplaats
0089 01		0137		00D6 6D	ADC	absolute	
008A 8D	STA	absolute	displaycode	00D7 1C		display	
008B 20		0120	in 'hoogte'	00D8 01		011C	
008C 01		0120		00D9 A8	TAY	implied	
008D C8	INY	implied	haal de	00DA B9	LDA	abs, Y	
008E B1	LDA	(ind), Y	toonlengte	00DB 00		0100	
008F EA		melody		00DC 01			displaycode naar het display
				00DD 8D	STA	absolute	
0090 EA	NOP	implied		00DE 40		PAD2	
0091 8D	STA	absolute	code voor de	00DF 17		1740	
0092 11		0111	lengte in 'delay'				
0093 01		0111		00E0 A0	LDY	immediate	wachtcyclus
0094 AD	LDA	absolute	code voor de	00E1 FF		\$FF	
0095 10		T	toon in A	00E2 88 loop 3	DEY	implied	
0096 01		0110		00E3 D0	BNE	relative	
0097 AA	TAX	implied	haal de	00E4 FD		loop 3	
0098 BD	LDA	abs, X	trillingstijd	00E5 68	PLA	implied	zet de registers weer goed, en spring terug
0099 00		tabel 3	uit de tabel	00E6 A8	TAY	implied	
009A 01		0100		00E7 68	PLY	implied	
009B 8D	STA	absolute	trillingstijd	00E8 60	RTS	implied	
009C 10		T	in 'T'				
009D 01		0110					
009E F0	BEQ	relative	is er een rust?				
009F 01		rust					

Lijst 2 Het interruptprogramma.

Interruptprogramma							
0140	48	PHA	implied	save A	014F	AD	LDA absolute
0141	AD	LDA	absolute		0150	12	octaaf
0142	10	T			0151	01	0112
0143	01		0110	set timer	0152	8D	STA absolute
0144	8D	STA	absolute	(8 µs)	0153	14	octaaf
0145	0D		timer		0154	01	0114
0146	17		170D		0155	EE	INC absolute
0147	CE	DEC	absolute		0156	00	PAD1
0148	14		octaaf	verminder octaaf	0157	17	1700
0149	01		0114	met 1	0158	68	return PLA implied
014A	D0	BNE	relative		0159	40	RTI implied
014B	0C		return	resultaat 0?			
014C	EE	INC	absolute				
014D	00		PAD1	Zo ja, inverteer			
014E	17		1700	uitgang			hoofdprogramma

Lijst 3 De door het programma gebruikte geheugenplaatsen.

De door het programma gebruikte geheugenplaatsen

00EA	00		beginadres van de melodie
00EB	02		paginanummer van de melodie
0100	00	tabel 3	rust
0101	EF	C	
0102	D5	D	
0103	BD	E	
0104	B3	F	
0105	9F	G	
0106	8E	A	deze tabel geeft
0107	7E	B	de trillingstijd van elke noot
0108	00	stop	uit het hoogste octaaf.
0109	E1	Cis	Andere getallen geven een
010A	C9	Dis	andere stemming
010B	B3	Eis	
010C	A9	Fis	
010D	96	Gis	
010E	86	Ais	
010F	77	Bis	
0110	T		deze geheugens worden
0111	delay		door het programma
0112	octaaf		gevuld met gegevens
0113			over de te spelen toon
0114	octaaf		
0115			
0116	5C	=	
0117	31	R	deze geheugens bevatten de
0118	79	E	displaycode voor het woord
0119	77	A	'READY', andere codes geven
011A	5E	D	andere letters.
011B	6E	Y	
011C	display		geeft aan welk blok van
011D			6 geheugens moet worden
011E			gedisplayd
011F			

0120	hoogte		displaycode van het octaaf	
0121	00		display 2 uit	
0122	toon		displaycode van de toon	
0123	i		'i' of uit	
0124	s		's' of uit	
0125	00		display 6 uit	
012F	00	tabel 1	uit	
0130	39	C	} displaycodes voor de naam van de noot. De juiste code wordt door het programma opgezocht en op 'toon' (0122) gezet.	
0131	5E	D		
0132	79	E		
0133	71	F		
0134	3D	G		
0135	77	A		
0136	7C	B		
0137	00	tabel 2	uit	} displaycodes voor de hoogte van het octaaf De juiste code wordt door het programma opgezocht en op 'hoogte' (0120) gezet.
0138	01	1		
0139	41	2		
013A	00	uit		
013B	48	4		
013C	00	uit		
013D	00	uit		
013E	00	uit		
013F	08	8		
1700	PAD1		Op PAD1 is de luidspreker en de schakelaar aangesloten	
1701	PADD1		Richtingsregister voor PAD1	
170D	timer		Ingestelde tijd = inhoud x 8 µs	
1740	PAD2		De inhoud van PAD2 komt op het display te staan	
1741	PADD2		Richtingsregister voor PAD2	
1742	PBD2		Bepaalt welke van de 6 displays oplicht	
1743	PBDD2		Richtingsregister voor PBD2	