ON A DIGITAL COMPUTER WHICH CAN BE PROGRAMMED IN A MATHEMATICAL FORMULA LANGUAGE

BY

L. KALMÁR /Szeged/

The description of a computer is sketched whose program language /as a matter of fact, a variant of the program language used in connection with Liapunov's operator method/ is close to usual mathematical formula language.

THE PROGRAM LANGUAGE. 1. Primitive symbols: 1,1. digits 0,1,...,9; 1,2. decimal comma; 1,3 number end symbol ¬ ; 1,4. simple variables /any letters which are wanted to be used as denotations of scalar quantities/; 1,5. array identifiers /any letters, different from simple

variables, which are wanted to be used as denotations of vectors, matrices etc./; 1,6. arithmetical parentheses (,) ; 1,7. subscript brackets [,]; 1,8. subscript delimiter , ; 1,9. arithmetical operation symbols +, -, ×, /, ↑ /for exponentiation/, $\sqrt{}$, * /for standard one argument elementary functions/; 1,10. relation symbols =, ≠, <, ≧, >, ≦ ; 1,11. logical connectives $\wedge, \bar{\wedge}, \vee, \bar{\vee}, \rightarrow, \not\rightarrow, \leftrightarrow, \not\leftrightarrow$; 1,12. storage symbols ⇒, ≻; 1,13. label delimiter ⊣ ; 1,14. jump symbols ⌜, ⌜*, ⌞ ; 1,15. loop braces {, }; 1,16. loop delimiters ⇐, ⟨, ⟩, : /colon/; 1,17. array size symbol ⊛; 1,18. printing symbol # ; 1,19. continuation symbol; /semicolon/; 1,20. stop symbol . /full stop/.

2. From these primitive symbols the following compound symbols are built: 2,1. /non-negative integer/ numbers, i.e. finite sequences of digits; 2,2. /non-negative/ constants of the form $\nu\urcorner$ or $\nu, \nu'\urcorner$ / ν, ν' denoting numbers/; 2,3. subscripted variables $\iota[\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_\lambda]$, ι denoting an array identifier and $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_\lambda$ expressions not containing subscripted variables. 2,4. A constant, a simple variable or a subscripted variable is called a term and at the same time, an expression; if τ_1 and τ_2 are terms and $\oplus$ is an arithmetical operation symbol, $(\tau_1 \oplus \tau_2)$ is a term and $\tau_1 \oplus \tau_2$ an expression; nothing else is a term nor an expression. 2,5. If ε_1 and ε_2 are expressions and Δ is a relation symbol, $(\varepsilon_1 \Delta \varepsilon_2)$ is a parenthesized formula and $\varepsilon_1 \Delta \varepsilon_2$ a formula; if φ_1 and φ_2 are parenthesized formulae and $\square$ is a logical connective, $(\varphi_1 \square \varphi_2)$ is a parenthesized formula and $\varphi_1 \square \varphi_2$ a formula; nothing else is a formula nor a parenthesized formula.

3. Out of expressions and formulae operators, array declarations, partial programs and programs are built as follows: 3,1. for any expression ε and any /simple or subscripted/ variable υ, $\varepsilon \Rightarrow \upsilon$; , $\varepsilon \Rightarrow \upsilon$. , $\varepsilon \Rightarrow$ # ; , $\varepsilon \Rightarrow$ # . , and . /the stop symbol alone/ are arithmetical operators, and nothing else is an arithmetical operator; 3,2. for any formula φ and any numbers ν and ν', $\varphi\nu\urcorner$, $\varphi\nu\lrcorner$, $\varphi\nu\urcorner^*\nu'\lrcorner$ and $\nu\urcorner$

/and nothing else/ are logical operators; 3,3. for any simple variable p, for any expressions $\varepsilon_1, \varepsilon_2, \varepsilon_3$, for any partial program π and any formula φ, $\{p \Leftarrow \varepsilon_1 \langle \varepsilon_2 \rangle \varepsilon_3 : \pi\}$ and $\{p \Leftarrow \varepsilon_1 \langle \varepsilon_2 \rangle : \pi\varphi\}$ /and nothing else/ are loop operators; 3,4. for any number ν and any /arithmetical, logical or loop/ operator ω, $\nu \sqcup \omega$ /and nothing else/ is called a labelled operator, ν its label /or that of ω / and ω is said to be labelled by ν; 3,5. for any expressions $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_\lambda$ and any array identifier ι, $\varepsilon_1 \otimes \varepsilon_2 \otimes \ldots \otimes \varepsilon_\lambda \succ \iota$ /and nothing else/ is called an array declaration; 3,6. any sequence of operators and/or labelled operators is called a partial program and also a program; any sequence of array declarators, operators and/or labelled operators is called a program; nothing else is a partial program or a program.

The /semantical/ meaning of the notions defined /syntactically/ thus, will be explained by the following remarks and examples. /2,2./ The symbol $\neg$, indicating end of a constant, will simplify some orders in the order list of the computer; at the cost of their complication, this symbol can be dispensed with. /2,3, 1,8./ If, e.g., A /the name of a vector/, B /the name of a matrix/ and C /the name of a three dimensional array/ are array identifiers and i, j, k are simple variables, say, then $A[i]$ denotes the i^{th} component /usually denoted by A_i / of A, $B[i, j]$ the element /usually denoted by B_{ij} / in the i^{th} row and j^{th} column of B and $C[i, j, k]$ the element in the i^{th} row, j^{th} column and k^{th} layer of C./2,4, 1,9./ For each standard one argument elementary function /besides exponentiation and root extraction as two argument functions/ a subroutine, identified by a distinction number, has to be built into the computer. The value of any standard one argument elementary function is regarded as the result of an operation, denoted by $*$, the left and right componente of which being the distinction number of the corresponding subroutine and the argument of the function, respectively. However, the use of the customary notations as sin x, e.g., is allowed, provided

that e.g. the word sin is coded in the same way as $(\nu\urcorner *$, ν denoting the distinction number of the subroutine for sin. /The lack of the right parenthesis does not matter./ /3,1./ The arithmetical operator $\varepsilon \Rightarrow \upsilon$; or $\varepsilon \Rightarrow \upsilon$. means: compute the value of ε /for the actual values of the variables figuring therein/, replace the value of υ by the value computed thus or give υ this value if it did not have a value previously, and proceed serially or stop, respectively; $\varepsilon \Rightarrow \#$; or $\varepsilon \Rightarrow \#$. means: compute and print the value of ε and proceed serially or stop, respectively; . means: stop. /3,2./ The logical operators $\varphi\nu\Gamma$ and $\varphi\nu\llcorner$ mean: compute the truth value of φ and jump to the operator labelled by ν if this truth value is "true" and if it is "false", respectively, otherwise proceed serially; $\varphi\nu\Gamma\nu'\llcorner$ means: compute the truth value of φ and jump to the operator labelled by ν or ν' according as this truth value is "true" or "false" respectively; $\nu\Gamma$ means: jump to the operator labelled by ν . /3,3./ The loop operators $\{p \Leftarrow \varepsilon_1\langle\varepsilon_2\rangle\varepsilon_3 : \pi\}$ and $\{p \Leftarrow \varepsilon_1\langle\varepsilon_2\rangle : \pi\varphi\}$ mean: perform the partial program π successively for the values $\varepsilon_1, \varepsilon_1+\varepsilon_2, \varepsilon_1+2\varepsilon_2, \cdots$ of the parameter p until its value does not exceed ε_2, and until φ is satisfied, respectively, and afterwards, proceed serially, $\varepsilon_1, \varepsilon_2$ and ε_3 denoting the values of these expressions before the first performing of π whereas in φ, each variable has to be meant to have its value after the actual performing of π. /3,5./ The array declaration $\varepsilon_1 \otimes \varepsilon_2 \otimes \cdots \otimes \varepsilon_\lambda \succ \iota$ means: ι is an array with /the value of/ $\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_\lambda$ as the number of its rows, columns etc.

In the case an array declaration contains a simple variable to which no value has been given by a previous operator, or a subscripted variable, to which no unique previous array declaration /with the same array identifier as that in front of the subscript bracket/ corresponds /"previous" meant always in dynamic sense/, or the values of its subscripts do not match with this array declaration, or a logical operator contains /in front of the symbol $\Gamma, \llcorner$ or Γ^* / a number which is not the label of an unique

operator, the operator or array declaration in question, and any partial program, loop operator or program containing it, has to be regarded as meaningless.

THE COMPUTER. Besides the number of possible simple variables, array identifiers and the total number of possible array elements, determined by the memory capacity of the computer, the following data are regarded as essential capacity data of the computer: /1/ the maximal number n of nested pairs of parentheses; /2/ a number m such that the numbers $0,1,2,\dots m$ and only these are allowed to be used as labels; /3/ the maximal dimension d of arrays; /4/ the maximal number k of nested pairs of loop braces.

The principial units of the computer contain the following sub-units and registers.

1. Input. /1/ A perforator /or the like/ for preparing the program tape. /2/ A tape reader transferring the code numbers of the successive primitive symbols of the program in the program tape into the memory unit as successive components $P[0], P[1], P[2], \dots$ of a "program vector".

2. Memory unit. /1/ A part of its cells, addressed by the code numbers of the simple variables, is reserved for storage of the /changing/ values of these variables. /2/ Another part of its cells, addressed by the code number of the array identifiers, is reserved for storage of informations, each packed into a single word, about the initial address and the address steps corresponding to the increase by 1 of the subscripts, of the corresponding array elements. /3/ The rest of the memory cells serves for storage of the /changing/ values of the array elements, the components of the program vector included.

3. Arithmetical unit. /1/ Computing units for performing the elementary arithmetical operations $(+, -, \times, /)$ as well as the operations corresponding to the elementary arithmetical relations $(=, \neq, <, \geqq, >, \leqq)$ and logical connectives $(\wedge, \bar{\wedge}, \vee, \bar{\vee}, \rightarrow, \not\rightarrow, \leftrightarrow, \not\leftrightarrow)$, further built in subroutines with automatical return to the program, for performing exponen-

tiation and root extraction as well as the standard one argument elementary functions. /As the operation corresponding to an arithmetical relation $\triangle$, or to a logical connective $\square$ any arithmetical operation may be chosen whose result, performed on a and b, is positive if and only if $a \triangle b$ or $(a>0) \square (b>0)$, resp. holds true. Each computing unit or built in subroutine has, according to the number of components of the corresponding operation, one or two input registers as well as an output register /for the result of the operation/. /2/ A converter for conversion of constants written in decimal form to machine form /binary, say/. It contains an output register COR, containing 0 in the initial state, as well as a conversion factor register CFR, containing 10 in the initial state and able to contain negative powers of 10 as well. /3/ $n+1$ arithmetical blocks $AB_0, AB_1, \dots, AB_n$ of four registers LOR_i, OpR_i, ROR_i and RR_i $(i = 0,1,\dots,n)$ each. Their initial or "empty" state must be different from any state in which they contain, or "are filled by" any real numbers /even 0/. If the registers LOR_i and ROR_i are filled by the components and OpR_i by the code number of the symbol of an operation $(+, -, \times, /, \uparrow, \sqrt{}, *,$ or that corresponding to $=, \neq, <, \geqq, >, \leqq, \wedge, \bar{\wedge}, \vee, \bar{\vee}, \rightarrow, \not\rightarrow, \leftrightarrow, \not\leftrightarrow)$ the contents of LOR_i and ROR_i are automatically transferred into the input registers of the corresponding computing unit or built in subroutine /in the case of the operation $*$, the content of ROR_i is transferred to the input register of the subroutine for a standard one argument elementary function having the content of LOR_i as distinction number/. After performing the required computation, the result is automatically transferred from the output register of the computing unit or subroutine into RR_i and from this register, provided $i > 0$, into that of LOR_{i-1} or ROR_{i-1}, by which previously RR_i has been "connected". At the same time, LOR_i, ROR_i, OpR_i and RR_i are cleared and the said connection broken. /4/ An address computing unit containing a free cell register FCR for keeping the address of the first memory cell available for storing array elements /containing in the initial state, after reading the program tape, the

address of the first memory cell for array elements, not taken up by the program vector/, further an address register AR as well as $d+1$ address step registers $ASR_0, ASR_1, \ldots, ASR_d$ and an output or information register IR. In the initial state, ASR_0 has to contain the number 1, whereas $ASR_1, \ldots, ASR_d$ the number 0.

4. Control unit. /1/ An order counter OC. /2/ An order register OR. /3/ Two control register CR and CR′ the first of which for keeping a distinction number of one of the registers $LOR_0, ROR_0, RR_0, LOR_1, ROR_1, RR_1, \ldots, LOR_n, ROR_n, RR_n, IR, PR_1, PR_2, \ldots, PR_k$ /in the initial state, of LOR_0 /, called the active register, whereas the second for keeping the content of CR while the latter is engaged in computing the values of subscripts of array components. /4/ A loop counter LC counting the number of loop operators performing of which has been started but not yet finished /containing 0 in the initial state/. /5/ $m+1$ jumping place register $JPR_0, JPR_1, \ldots, JPR_m$ for keeping the addresses of the code numbers of the first symbols /as components of the program vector/ of the operators labelled by $0, 1, \ldots, m$, respectively. /6/ k parameter registers $PR_1, PR_2, \ldots, PR_k$ for keeping the code numbers of the parameters of loop operators; /7/ k parameter step register $PSR_1, PSR_2, \ldots, PSR_k$; /8/ k final value registers $FVR_1, FVR_2, \ldots, FVR_k$; /9/ k loop start registers $LSR_1, LSR_2, \ldots, LSR_k$. Provided, LC contains i $(= 1, 2, \ldots, k)$, PR_i is called the working parameter register.

5. Output. /1/ A printing register PrR for keeping the real number to be printed until its being re-converted into decimal form and printed. /2/ A re-converter for conversion of the content of the PrR from machine form into decimal form. /3/ A teleprinter /or the like/ for printing decimal /real/ numbers.

The order list of the computer. After typing the program tape on the perforator, inserting it into the tape reader and pressing the start button, the code numbers of the successive symbols of the program are transferred into the memory, otherwise all memory cells and registers are cleared un-

less their initial state has been given above in which case they go over into the initial state. Moreover, the information word indicating the initial address as well as the unique address step 1 of the program vector /the others being 0/ is stored into the memory cell addressed with the code number of the array identifier P used as the name of the program vector and the computer starts to work with the initial address of the program vector as content of the order counter. In each working cycle of the computer, first the content of the memory cell addressed with the content of the order counter is transferred into the order register, then the content of the order counter is increased by one, finally the order corresponding to the content of the order register is performed according to the following order list, after which the next working cycle begins. In stating the order list, the content of a register is denoted by putting its name into parentheses, and the content of a memory cell addressed by the content of a register is denoted by putting the name of the latter into double parentheses, further usual mathematical notation as well as the notation $\Rightarrow$ is used.

Order list

Primitive symbol the code number of which equals (OR)	Condition	Order to be performed
A digit δ	/1/ (CFR) $=$ 10	$(CFR)(COR)+\delta \Rightarrow (COR)$
	/2/ (CFR) $\neq$ 10	$(COR)+\delta(CFR) \Rightarrow (COR)$, afterwards $0,1\,(CFR) \Rightarrow (CFR)$
A decimal comma,		$0,1 \Rightarrow (CFR)$
A number end symbol ¬	/1/ a LOR_i active	$(COR) \Rightarrow (LOR_i)$, $10 \Rightarrow (CFR)$ afterwards $0 \Rightarrow (COR)$ and activate OpR_i
	/2/ an ROR_i active, CR′ empty	$(COR) \Rightarrow (ROR_i)$, $10 \Rightarrow (CFR)$,

Primitive symbol the code number of which equals (OR)	Condition	Order to be performed
		afterwards 0 ⇒ (COR), and activate the OpR_j with maximal $j<i$ for which OpR_j empty, or, if such a j does not exist, RR_0
	/3/ an ROR_i active, (CR′) equals the distinction number of LOR_p or ROR_p	(COR) ⇒ (ROR_i), 10 ⇒ (CFR), afterwards 0 ⇒ (COR), and activate the OpR_j with maximal $p<j<i$ for which OpR_j empty, or, if such a j does not exist, RR_{p+1}
A simple variable	/1/ a LOR_i active	((OR)) ⇒ (LOR_i) and activate OpR_i
	/2/ an ROR_i active, CR″ empty	((OR)) ⇒ (ROR_i) and activate the OpR_j with maximal $j<i$ for which OpR_j empty, or, if such a j does not exist, RR_0
	/3/ an ROR_i active, (CR′) equals the distinction number of LOR_p or ROR_p	((OR)) ⇒ (ROR_i) and activate the OpR_j with maximal $p<j<i$ for which OpR_j

Primitive symbol the code number of which equals (OR)	Condition	Order to be performed
		empty, or, if such a j does not exist, RR_{p+1}
	/4/ RR_0 active	$(RR_0) \Rightarrow ((OR))$, clear RR_0 /i.e. put it into the initial state/ and activate LOR_0
	/5/ a PR_i active	$(OR) \Rightarrow (PR_i)$ and activate LOR_0
An array identifier	/1/ IR active	$(IR) \Rightarrow ((OR))$ and activate LOR_0
	/2/ a LOR_i or ROR_i or RR_i $(i = 0)$ active	unpack $((OR))$, considered as information word, into an initial address and d address steps, $\Rightarrow (AR), (ASR_1), (ASR_2), \ldots, (ASR_d)$, afterwards $(CR) \Rightarrow (CR')$ and activate LOR_{i+1}
A left parenthesis (	a LOR_i or ROR_i active	connect the active register with RR_{i+1} and activate LOR_{i+1}
A right parenthesis)		no operation
A left bracket [		no operation
A right bracket]	/1/ an OpR_i active	code number of + $\Rightarrow (OpR_i)$, $0 \Rightarrow (ROR_i)$, afterwards as in /2/
	/2/ an RR_i active	$(AR) + (RR_i)(ASR_1) \Rightarrow (OR)$, afterwards $(CR') \Rightarrow (CR)$, afterwards clear CR',

Primitive symbol the code number of which equals (OR)	Condition	Order to be performed
		afterwards as in the case of a simple variable
A subscript deliniter ,	/1/ an OpR_i active	code number of + ⇒ (OpR_i) 0 ⇒ (ROR_i) , afterwards as in /2/
	/2/ an RR_i active	$(AR)+(RR_i)(ASR_1) \Rightarrow (AR)$, afterwards successively $(ASR_2) \Rightarrow \Rightarrow (ASR_1),\ldots,(ASR_d) \Rightarrow (ASR_{d-1})$, $0 \Rightarrow (ASR_d)$ and activate LOR_i
An arithmetical operation symbol, relation symbol or logical connective	an OpR_i active	$(OR)\ (Op_i)$, activate ROR_i
A storage symbol ⇒	/1/ OpR_0 active	code number of + ⇒ (OpR_0), $0 \Rightarrow (ROR_0)$, afterwards as in /2/
	/2/ RR_0 active	clear LOR_0, OpR_0 and ROR_0
A storage symbol ≻	/1/ OpR_0 active	code number of + ⇒ (OpR_0), $0 \Rightarrow (ROR_0)$, afterwards as in /2/
	/2/ RR_0 active, $(ASR_i) = 1$, $(ASR_{i+1}) = 0$	$(RR_0)(ASR_j) \Rightarrow (ASR_j)$ for $j = 0,1,\ldots,i$, $1 \Rightarrow (ASR_{i+1})$, afterwards clear LOR_0, OpR_0, ROR_0 and RR_0 , pack (FCR) as initial address and $(ASR_1),\ldots,(ASR_d)$ as address steps into an inform-

Primitive symbol the code number of which equals (OR)	Condition	Order to be performed
		ation word, ⇒ (IR), afterwards (FCR)+(ASR$_0$)⇒(FCR), afterwards 1 ⇒ ⇒ (ASR$_0$) and 0 ⇒ (ASR$_j$) for j = 1,2,...,d, and activate IR
A label delimiter ⌋	(COR) = i	(OC)⇒(JPR$_i$), 10⇒(CFR), 0⇒(COR)
A jump symbol ⌜	/1/ (COR) = i, RR$_0$ empty or (RR$_0$) > 0	(JPR$_i$)⇒(OC), 10⇒(CFR), 0⇒(COR), clear LOR$_0$, OpR$_0$, ROR$_0$ and RR$_0$ and activate LOR$_0$
	/2/ (RR$_0$) ≦ 0	10⇒(CFR), 0⇒(COR), clear LOR$_0$, OpR$_0$, ROR$_0$ and RR$_0$ and activate LOR$_0$
A jump symbol ⌜*	/1/ (COR) = i (RR$_0$) > 0	as in the case of a jump symbol ⌜
	/2/ (RR$_0$) ≦ 0	10 ⇒ (CFR), 0 ⇒ (COR)
A jump symbol ⌞	/1/ (COR) = i, (RR$_0$) ≦ 0	as in the case /1/ of a jump symbol ⌜
	/2/ (RR$_0$) > 0	as in the case /2/ of a jump symbol ⌜
A left brace {		(LC)+1 ⇒ (LC), afterwards activate the working parameter register
A right brace }	/1/ LOR$_0$ active, (LC) = i, ((PR$_i$)) + +(PSR$_i$) ≦ ≦ (FVR$_i$)	((PR$_i$))+(PSR$_i$) ⇒ ((PR$_i$)), (LSR$_i$) ⇒ (OC)

Primitive symbol the code number of which equals (OR)	Condition	Order to be performed
	/2/ LOR_0 active, $(LC) = i$, $((PR_i)) + (PSR_i) > (FVR_i)$	$(LC) - 1 \Rightarrow (LC)$
	/3/ RR_0 active, $(RR_0) > 0$, $(LC) = i$	as in /1/, afterwards clear LOR_0, OpR_0, ROR_0 and RR_0 and activate LOR_0
	/4/ RR_0 active, $(RR_0) \leqq 0$	as in /2/, afterwards clear LOR_0, OpR_0, ROR_0 and RR_0 and activate LOR_0
A loop delimiter $\Leftarrow$		no operation
A loop delimiter $\langle$	/1/ OpR_0 active, $(LC) = i$	code number of $+ \Rightarrow (OpR_0)$, $0 \Rightarrow (ROR_0)$, afterwards as in /2/
	/2/ RR_0 active, $(LC) = i$	$(RR_0) \Rightarrow ((PR_i))$, clear LOR_0, OpR_0, ROR_0 and RR_0 and activate LOR_0
A loop delimiter $\rangle$		as in the case of a loop delimiter $\langle$ however, with $(RR_0) \Rightarrow (PSR_i)$ instead of $(RR_0) \Rightarrow ((PR_i))$
A loop delimiter:		as in the case of a loop delimiter $\langle$, however,

Primitive symbol the code number of which equals (OR)	Condition	Order to be performed
		with $(RR_0) \Rightarrow (FVR_i)$ instead of $(RR_0) \Rightarrow ((PR_i))$, further, $(OC) \Rightarrow (LSR_i)$
An array size symbol ⊗	/1/ OpR_0 active	code number of + $\Rightarrow (OpR_0)$, $0 \Rightarrow (ROR_0)$, afterwards as in /2/
	/2/ RR_0 active, $(ASR_i) = 1$, $(ASR_{i+1}) = 0$	$(RR_0)(ASR_j) \Rightarrow (ASR_j)$ for $j = 0, 1, \ldots, i$, $1 \Rightarrow (ASR_{i+1})$, afterwards clear LOR_0, OpR_0, ROR_0 and RR_0 and activate LOR_0
A printing symbol #		$(RR_0) \Rightarrow (PrR)$, afterwards clear RR_0 and activate LOR_0 . During continuation of performing the program, (PrR) will be re-converted into decimal form and printed.
A continuation symbol ;		no operation
A stop symbol		stop

By slight modifications of the order list, some convenient changes of the program language can be made possible. E.g. for any term τ, $(+\tau)$ and $(-\tau)$ can be regarded as terms and $+\tau$ and $-\tau$ as expressions provided the order list in the case if (OR) equals the code number of the symbol + or - will be appropriately changed .