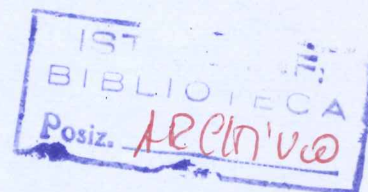
Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA



THE SERIAL MICROPROCESSOR ARRAY (SMA): MICROPROGRAMMING AND APPLICATION EXAMPLES

P. Corsini - G. Frosini - F. Grandoni -
G. Galati - M. La Manna

L. 78 - 4

Estratto da: Conference Proceedings of
The 5th Annual Symposium on
Computer Architecture -
April 3-5, 1978 - pp. 230-235

THE SERIAL MICROPROCESSOR ARRAY (SMA): MICROPROGRAMMING AND APPLICATION EXAMPLES

P. Corsini*, G. Frosini**, F. Grandoni***, G. Galati***, M. La Manna***

*Dipartimento Sperimentale di Elettrotecnica ed Elettronica, Università di Pisa, Pisa, Italy.
**Istituto di Elaborazione della Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy.
***Selenia, Industrie Elettroniche Associate, S.p.A., Roma, Italy.

The structure of the Processing Element (PE), which is the basic component of SMA[1], is presented. The PE consists of a simple serial arithmetic unit, a local high speed data memory, serial input and output ports, serial communication channels with neighbouring PE's, and some local control logic. The PE array operates under the control of a microprogrammed Array Control Unit (ACU). The peculiarities of ACU microprogramming are discussed, and some typical microprograms are reported. After presentation of the SMA principal instructions, some application programs are described implementing common radar filtering algorithms.

1. Introduction

The high data rate and the highly regular structure of data to be processed in radar systems have long been a stimulating challenge to computer designers. Processing tasks in radar systems can be classified in two levels: (1) digital filtering algorithms needed to extract target information from the raw video signal; (2) target tracking, e.g. for collision prevention in air traffic control systems. Computing structures oriented towards second level radar data processing have been studied to a remarkable extent[2-7]. The first level, more properly referred to as radar signal processing, involves many tens of arithmetic operations per sample, sampling interval of input signal being of the order of 0.25 to 1.0 μsec. These throughput figures required the use of special-purpose digital video processors, where the filtering algorithms were executed by hardwired logic. In an attempt to overcome the drawbacks exhibited by hardwired processors, several architectural proposals were presented[8-10]. A research project on a programmable radar signal processor sponsored by the Selenia-CNR Convention led to the design of Serial Microprocessor Array (SMA)[1].

The SMA is an array processor constituted by a large number of Processing Elements (PE's), typically one thousand or more. The characteristics of radar data and radar processing algorithms, which justify the architecture of SMA, are examined in ref [1]. In this paper we concern ourselves with details of PE structure, with array microprogramming, and with some programming examples.

The structure of the SMA is given in Fig. 1. It consists of two subsystems: the Master Computer and the Slave Computer. The Master Computer (MC) controls the program flow and sends for execution to the Slave Computer the instructions that operate on radar data. (array instructions). The MC is constituted by a Main Memory, that stores the program and those data that are common to all the Processing Elements, and by a Processing Unit (CPU), with eight 16-bit general registers and limited arithmetic and logic capabilities.

The Slave Computer (SC) is constituted by an array of Processing Elements, each endowed with its own memory, and by a microprogrammed Array Control Unit (ACU). The control unit accepts array instructions from the Master Computer via an Asynchronous FIFO Buffer (AFB)

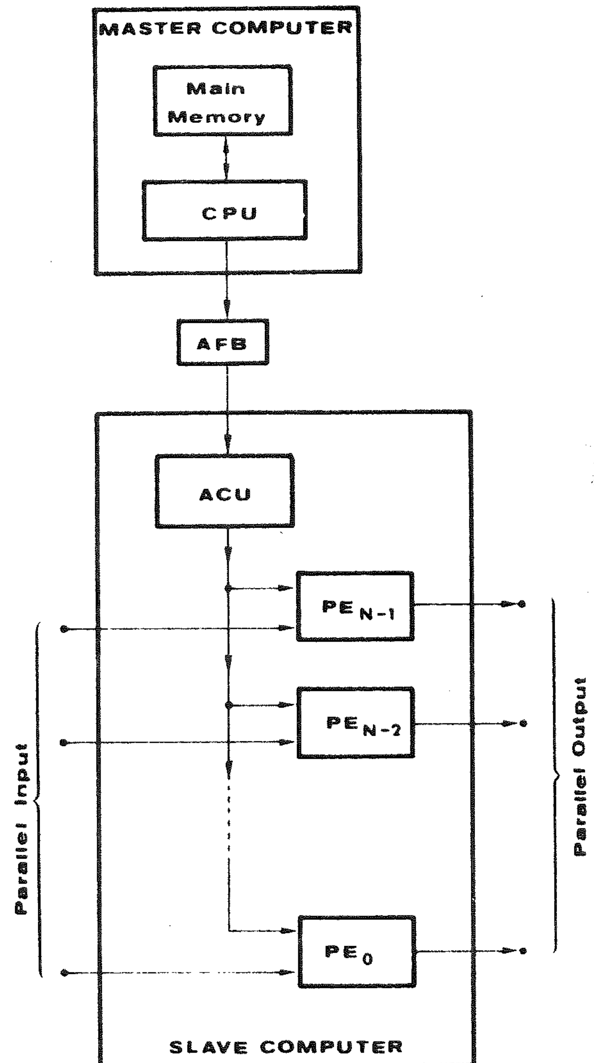


Fig. 1. System organization

This work has been sponsored by the Convention between Selenia S.p.A. and Consiglio Nazionale delle Ricerche, Pisa, Italy.

and sends microorders in parallel to all microprocessors. Serial arithmetic on variable length operands is implemented, and fractional two's complement number representation is used. Operand length is from 1 to 16 bits.

2. Set of instructions

The program instructions can be distinguished into two classes:

1) instructions that control the program flow;
2) instructions that perform arithmetic and logic operations on array data (array instructions).

The instructions of the first class are similar to those of a typical minicomputer, so they are not worth to note. Their execution is accomplished directly by the MC. The MC is also assigned a less conventional task: it fetches the array instructions, computes the physical addresses of operands in PE memories, assembles in a register all the information needed by the ACU for the instruction execution. At last, the packed information is dispatched through the AFB to the ACU. To carry out these functions the MC is equipped with eight 16-bit general registers, five of which are assigned the following specific tasks:

- R0 : base register;
- R1 : index register;
- R2 : register holding an operand common to all PE's;
- R3,R4: registers holding the operand lengths (in the rightmost four bits).

The array instructions are executed simultaneously by the PE's of the Slave Computer. They can be partitioned into three subclasses:

a) arithmetic instructions;
b) logical instructions and test instructions;
c) input/output instructions.

The formats of the three subclasses are shown respectively in Figs. 2a, 2b and 2c. The OP field represents

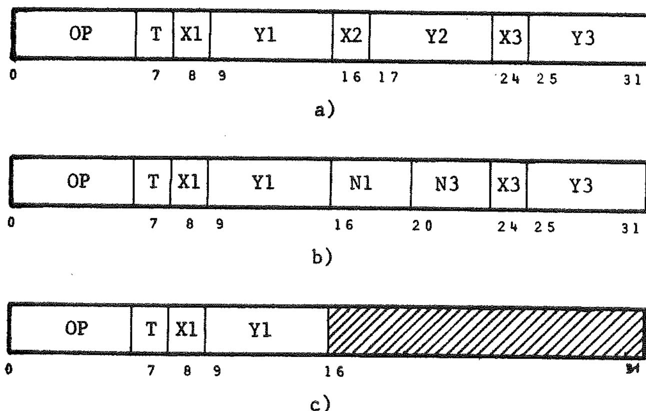


Fig. 2 a) Arithmetic instruction format
b) Logical and test instruction format
c) Input/Output instruction format

the operation code and the pair (Xi, Yi), i=1,2,3, specifies the logical address of one operand stored in the PE memories, each memory thought of as arranged in 16-bit words. The logical address is given by Yi if Xi=0, and it is obtained by adding Yi to the contents of register R1, if Xi=1. The logical address is mapped into a physical address Fi by adding to it the contents of register R0 of the MC. This relocation mechanism is used to implement a sliding working area, as shown in Section 5. The T bit specifies if the instruction is to be executed in parallel either by all PE's (T=1) or only by those PE's having the TAG register set to 1 (T=0) (see Section 3). The fields N1 and N3 have a meaning depending on the specific instruction.

The following are typical instructions of the second class. The notation $M_\alpha(Fi)_{0-\beta}$ indicates the leftmost $\beta+1$ bits of the contents of the Fi-th word of M_α, where M_α is the memory of α-th PE, and $\beta \in \{0,1,\dots,15\}$. Similarly, $R2_{0-\beta}$ indicates the leftmost $\beta+1$ bits of the contents of register R2 of che CPU. Moreover, L1 and L2 denote the contents of the rightmost four bits of registers R3 and R4 of the CPU, respectively.

a) Arithmetic instructions.

1) ADD (add)
$M_n(F1)_{0-L1}$ added to $M_n(F2)_{0-L1}$ replaces $M_n(F3)_{0-L1}$;

2) SB (subtract)
$M_n(F2)_{0-L1}$ subtracted from $M_n(F1)_{0-L1}$ replaces $M_n(F3)_{0-L1}$;

3) ADC (add constant)
$M_n(F1)_{0-L1}$ added to $R2_{0-L1}$ replaces $M_n(F3)_{0-L1}$;

4) SBC (subtract constant)
$R2_{0-L1}$ subtracted from $M_n(F1)_{0-L1}$ replaces $M_n(F3)_{0-L1}$;

5) ADU1 (add 1st upper)
$M_n(F1)_{0-L1}$ added to $M_{n+1}(F2)_{0-L1}$ replaces $M_n(F3)_{0-L1}$;

6) SBU1 (subtract 1st upper)
$M_{n+1}(F2)_{0-L1}$ subtracted from $M_n(F1)_{0-L1}$ replaces $M_n(F3)_{0-L1}$;

7) ADD1 (add 1st down)
$M_n(F1)_{0-L1}$ added to $M_{n-1}(F2)_{0-L1}$ replaces $M_n(F3)_{0-L1}$;

8) SBD1 (subtract 1st down)
$M_{n-1}(F2)_{0-L1}$ subtracted from $M_n(F1)_{0-L1}$ replaces $M_n(F3)_{0-L1}$;

9) MADU3 (move and add 3rd upper)
$M_{n+3}(F1)_{0-L1}$ repalces $M_n(F3)_{0-L1}$, and $M_{n+3}(F1)_{0-L1}$ added to $M_n(F2)_{0-L1}$ replaces $M_n(F2)_{0-L1}$;

10) MSBU3 (move and subtract 3rd upper)
$M_{n+3}(F1)_{0-L1}$ replaces $M_n(F3)_{0-L1}$, and $M_{n+3}(F1)_{0-L1}$ subtracted from $M_n(F2)_{0-L1}$ replaces $M_n(F2)_{0-L1}$;

11) MADD3 (move and add 3rd down)
$M_{n-3}(F1)_{0-L1}$ replaces $M_n(F3)_{0-L1}$, and $M_{n-3}(F1)_{0-L1}$ added to $M_n(F2)_{0-L1}$ replaces $M_n(F2)_{0-L1}$;

12) MSBD3 (move and subtract 3rd down)
$M_{n-3}(F1)_{0-L1}$ replaces $M_n(F3)_{0-L1}$, and $M_{n-3}(F1)_{0-L1}$ subtracted from $M_n(F2)_{0-L1}$ replaces $M_n(F2)_{0-L1}$;

13) MUL (multiply)
$M_n(F1)_{0-L1}$ multiplied by $M_n(F2)_{0-L2}$ replaces $M_n(F3)_{0-L1}$;

14) MULC (multiply by constant)
$M_n(F1)_{0-L1}$ multiplied by $R2_{0-L2}$ replaces $M_n(F3)_{0-L1}$;

15) DIV (divide)
$M_n(F1)_{0-L1}$ divided by $M_n(F2)_{0-L1}$ replaces $M_n(F3)_{0-L2}$;

16) TRAN (transfer from a memory location to another)
$M_n(F1)_{0-L1}$ replaces $M_n(F3)_{0-L1}$.

b) Logical instructions and test instructions.

1) SHL (shift left)
$M_n(F1)_{0-L1}$ left shifted arithmetically N1 times replaces $M_n(F3)_{0-L1}$;

2) SHR (shift right)
$M_n(F1)_{0-L1}$ right shifted arithmetically N1 times replaces $M_n(F3)_{0-L1}$

3) TQ (test and quantize)
if $M_n(F1)_{0-L1}$ is greater than or equal to $M_n(F3)_{0-L1}$, then 1 replaces the (N3+1)th bit of $M_n(F3)$, else 0 replaces the (N3+1)th bit of $M_n(F3)$;

4) TCQ (test against constant and quantize)
if $M_n(F1)_{0-L1}$ is greater than or equal to $R2_{0-L1}$, then 1 replaces the (N3+1)th bit of $M_n(F3)$, else 0 replaces the (N3+1)the bit of $M_n(F3)$;

5) TST (test and set TAG)
if $M_n(F1)_{0-L1}$ is greater than or equal to $M_n(F2)_{0-L1}$, then 1 replaces the contents of TAG register, else 0 replaces the contens of TAG register;

6) TRT (test and reset TAG)
if $M_n(F1)_{0-L1}$ is greater than or equal to $M_n(F2)_{0-L1}$, than 0 replaces the contents of TAG register, else 1 replaces the contents of TAG register

7) TCST (test against constant and set TAG)
if $M_n(F1)_{0-L1}$ is greater than or equal to $R2_{0-L1}$, then 1 replaces the contents of TAG register, else 0 replaces the contents of TAG register;

8) TCRT (test against constant and reset TAG)
if $M_n(F1)_{0-L1}$ is greater than or equal to $R2_{0-L1}$, then 0 replaces the contents of TAG register, else 1 replaces the contents of TAG register;

9) LOT (load TAG)
the (N1+1)th bit of $M_n(F1)$ replaces the contents of TAG register;

10) COT (complement TAG)
the complement of the contents of TAG register replaces the contents of TAG register;

11) ANDB (and bit)
the logical AND between the (N1+1)th bit of $M_n(F1)$ and the (N3+1)th bit of $M_n(F3)$ replaces the (N3+1)th bit of $M_n(F3)$;

12) ORB (or bit)
the logical inclusive-OR between the (N1+1)th bit of $M_n(F1)$ and the (N3+1)th bit of $M_n(F3)$ replaces the (N3+1)th bit of $M_n(F3)$;

13) CMB (complement bit)
the complement of the(N1+1)th bit of $M_n(F1)$ replaces the (N3+1)th bit of $M_n(F3)$.

The AFB word contains all the information needed by the Slave Computer for the execution of the array instructions, i.e., the operation code OP, the T bit. the physical addresses of the operands F1, F2 and F3, the contents of register R2 (the constant CO) and the operands lengths (L1 and L2). F2 can be replaced by N1 and N3. The word of AFB has the format shown in Fig. 3.

The ACU gets an AFB word and stores the various fields in dedicated registers, designated SOP, ST, SF1, SF2, SF3, SCO, SL1, SL2. The ACU also contains the following registers, relevant to the microprogrammer:

I,J (4 bits) : used as counters in microprograms;
MAR (11 bits): holding the memory address;
RW (1 bit) controlling the operation mode of memory: 0 for reading, 1 for writing.

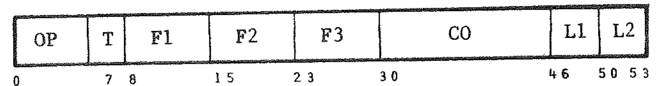


Fig. 3. AFB word format

3. Structure of processing element

The Processing Element (PE) is constituted by a high speed memory having up to 2048 bits and by a serial processing unit, both having the same cycle time.

The processing unit (Fig. 4) implements serial arithmetic using one 1-bit full adder/subtractor, six 1-bit registers with auxiliary logic, and the tag logic. The operation of the full adder/subtractor is controlled by the contents of register Z, which can be loaded by the ACU or directly by the internal PE logic, e.g., in the execution of the division instruction.The Register R holds the carry generated in the preceding step of addition or subtraction algorithms; its initial value is set by ACU. The operands come to the ALU, through the select network KN, from several sources: the PE's own memory, four neighbouring PE's, and the SCO register of ACU(which broadcasts a constant operand to all PE's).Register Y controls the network KN during the execution of the multiplication instruction.

Data bits from memory are temporarily held in the X,L registers. Register X also feeds four neighbouring PE's. The result bit from the ALU is stored back in memory via the A register.

The tag logic can block the PE operation, according to the result of a preceding test on local data. The test result is stored in a 1-bit TAG register; if the contents of TAG register is 0 the attendant logic disables the PE clock so that only the enabled PE's actually execute the microorders issued by the ACU.

4. Microprogramming the SMA

The array instructions, due to the serial arithmetic implemented and to the absence of full length registers in the PE, are memory-to-memory three-addresses instructions.

Every operating cycle requires several subsequent steps: 1st operand address loading in MAR, read memory cycle, 2nd operand address loading in MAR, read memory cycle, execute cycle, result address loading in MAR, write memory cycle. To obtain maximum speed it is necessary to overlap these steps in writing microprograms. The bottleneck being memory access, an operating cycle cannot be shorter than three clock cycles. In writing microprograms, MAR loading, memory read/write, and possibly ALU operation are pipelined so that in almost all microcycles the memory is kept busy. A register--transfer-type language is used in the description of microprograms. Each microinstruction is composed by a number of operation words, which specify the register transfers to be executed in the current cycle, and one or two jump words, which specify the next microinstruction to be executed. Only two alternative next microinstruction addresses are allowed,as this proved to be sufficient for the implementation of microprograms without undue delays.

In the presentation of some typical SMA microprograms the following conventions are used (besides that introduced in Section 2) with reference to a generic Processing Element:

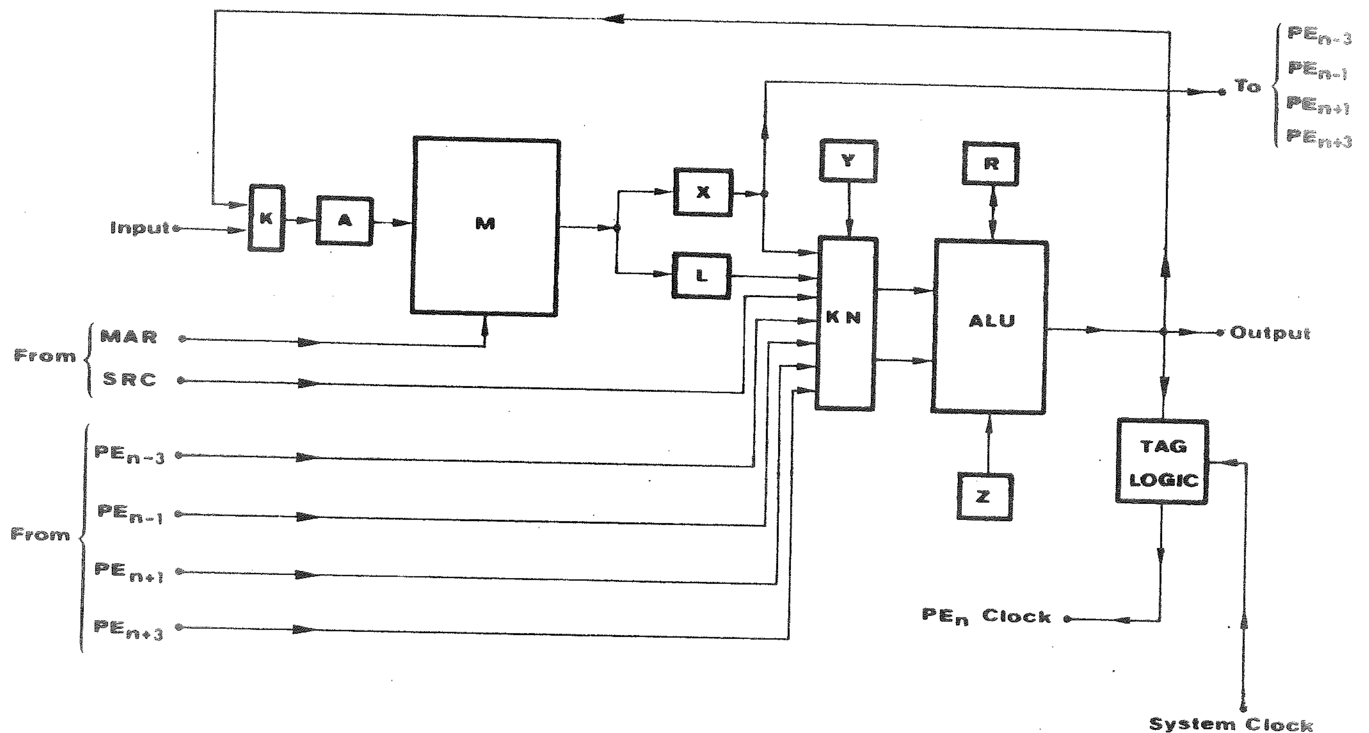


Fig. 4. The n-th Processing Element PE_n

L*(X ∧ Y): indicates the output of ALU (Fig. 4); the asterisk represents the ALU operation, i.e. summation if register Z is set to 0, subtraction otherwise; the operands are the bit stored in L and the logical AND of the contents of the registers X and Y;

r : is the carry output from ALU;

hC : is the first microinstruction of the microprogram which fetches a new instruction from the AFB;

IR,IL : are fixed words in memory M (whose addresses are hardwired in ACU) reserved as microprogram working areas;

ZERO : is a four grounded lines "register";

(A,B) : denotes two cascaded registers A and B taken as a single register.

Typical microprograms

AD $M(F1)_{0-L1}+M(F2)_{0-L1} \rightarrow M(F3)_{0-L1}$

h1	SL1 → I, 1 → Y, 0 → Z, 0 → R, h1 → h2
h2	(SF1,I) → MAR, h2 → h3
h3	I-1 → I, (SF2,I) → MAR, M(MAR) → L, h3 → h4
h4	I+1 → I, (SF1,I) → MAR, M(MAR) → X, h4 → h5
h5	I-1 → I, 1 → RW, (SF3,I) → MAR, M(MAR) → L, L*(X ∧ Y) → A, r → R, h5 → h6
h6	I-1 → I, 0 → RW, (SF2,I) → MAR, A → M(MAR), (I ≥ 0)h6 → h4, (I < 0)h6 → hC

MUL $M(F1)_{0-L1} \cdot M(F2)_{0-L2} \rightarrow M(F3)_{0-L1}$

h1	SL1 → I, SL2 → J, 0 → Z, 0 → R, h1 → h2
h2	I-1 → I, (SF2,J) → MAR, h2 → h3
h3	I-1 → I, (SF1,I) → MAR, M(MAR) → L, h3 → h4
h4	I+2 → I, (SF1,I) → MAR, M(MAR) → X, L → Y, (I ≥ 0)h4 → h5, (I < 0)h4 → h7
h5	I-3 → I, 1 → RW, (IR,I) → MAR, M(MAR) → X, 0*(X ∧ Y) → A, r → R, h5 → h6
h6	I+2 → I, 0 → RW, (SF1,I) → MAR, A → M(MAR), (I ≥ 0)h6 → h5, (I < 0)h6 → h7
h7	I-1 → I, 1 → RW, (IR,I) → MAR, 0*(X ∧ Y) → A, r → R, h7 → h8
h8	SL1 → I, J-1 → J, (IR,I) → MAR, A → M(MAR), h8 → h9
h9	0 → RW, 0 → R, (SF2,J) → MAR, A → M(MAR), h9 → h10
h10	(SF1,I) → MAR, M(MAR) → L, h10 → h11
h11	I-1 → I, (IR,I) → MAR, M(MAR) → X, L → Y, (J > 0)h11 → h12, (J ≤ 0)h11 → h19
h12	(SF1,I) → MAR, M(MAR) → L, h12 → h13
h13	I-1 → I, (IR,I) → MAR, M(MAR) → X, L*(X ∧ Y) → A, r → R, h13 → h14
h14	I+2 → I, (SF1,I) → MAR, M(MAR) → L, (I ≥ 0)h14 → h15, (I < 0)h14 → h17
h15	I-2 → I, 1 → RW, (IR,I) → MAR, M(MAR) → X, L*(X ∧ Y) → A, r → R, h15 → h16
h16	I-1 → I, 0 → RW, (IR,I) → MAR, A → M(MAR), h16 → h14
h17	I-1 → I, 1 → RW, (IR,I) → MAR, L*(X ∧ Y) → A, r → R, h17 → h18
h18	SL1 → I, J-1 → J, (IR,I) → MAR, L*(X ∧ Y) → A, A → M(MAR), h18 → h9
h19	I+1 → I, 1 → Z, (SF1,I) → MAR, M(MAR) → L, h19 → h20
h20	I-1 → I, 1 → RW, (SF3,I) → MAR, M(MAR) → X, L*(X ∧ Y) → A, r → R, h20 → h21
h21	I-1 → I, 0 → RW, (IR,I) → MAR, A → M(MAR), (I ≥ 0)h21 → h19, (I < 0)h21 → hC

DIV $M(F1)_{0-L1} : M(F2)_{0-L1} \rightarrow M(F3)_{0-L2}$

h1	0 → J, SL1 → I, 1 → R, 1 → Y, 0 → Z, (SF1,ZERO) → MAR, h1 → h2
h2	(SF2,ZERO) → MAR, M(MAR) → L, h2 → h3
h3	(SF2,I) → MAR, M(MAR) → X, h3 → h4
h4	1 → RW, (IR,J) → MAR, M(MAR) → X, L*(X ∧ Y) → A, 0 → R, L*(X ∧ Y) → Z, h4 → h5
h5	(IL,I) → MAR, A → M(MAR), 0*(X ∧ Y) → A, r → R, h5 → h6
h6	I-1 → I, 0 → RW, (SF1,I) → MAR, A → M(MAR), h6 → h7
h7	(SF2,I) → MAR, M(MAR) → L, h7 → h8
h8	(SF1,I) → MAR, M(MAR) → X, h8 → h9
h9	I-1 → I, 1 → RW, (IL,I) → MAR, M(MAR) → L, L*(X ∧ Y) → A, r → R, h9 → h10
h10	0 → RW, (SF2,I) → MAR, A → M(MAR), (I ≤ 0)h10 → h11, (I > 0)h10 → h8
h11	(IL,I) → MAR, M(MAR) → X, h11 → h12
h12	1 → RW, L*(X ∧ Y) → A, r → R, h12 → h13
h13	0 → RW, SL1 → I, A → M(MAR), h13 → h14

h14	J+1 → J, (SF2,I) → MAR, M(MAR) → L, 1 → R, (J ≥ SL2-1)h14 → h22, (J < SL2-1)h14 → h15
h15	1 → RW, (IR,J) → MAR, M(MAR) → X, L*(X ∧ Y) → A, 0 → R, L*(X ∧ Y) → Z, h15 → h16
h16	(IL,I) → MAR, A → M(MAR), 0*(X ∧ Y) → A, r → R, h16 → h17
h17	I-1 → I, 0 → RW, A → M(MAR), h17 → h18
h18	(SF2,I) → MAR, M(MAR) → L, h18 → h19
h19	(IL,I) → MAR, M(MAR) → X, h19 → h20
h20	I-1 → I, 1 → RW, M(MAR) → L, L*(X∧Y) → A, r→R, h20 → h21
h21	0→ RW, (SF2,I)→ MAR, A → M(MAR), (I ≤ 0)h21 → h11, (I > 0)h21 → h19
h22	J-1 → J, 1 → RW, (SF3,J) → MAR, L*(X ∧ Y) → A, 0 → R, h22 → h23
h23	J-1 → J, 0 →RW, 0 →Y, (IR,J)→ MAR, A →M(MAR), h23 → h24
h24	J+1 → J, (IR,J) → MAR, M(MAR) → L, h24 → h25
h25	J-2 → J, 1 → RW, (SF3,J) → MAR, M(MAR) → L, L*(X ∧ Y) → A, r → R, h25 → h26
h26	J+1 → J, 0 → RW, (IR,J) → MAR, A → M(MAR), (J ≤ 0)h26 → h27, (J > 0)h26 → h25
h27	J-1 → J, 1 → RW, (SF3,J) → MAR, M(MAR) → L, L*(X ∧ Y) → A, 1 → R, h27 → h28
h28	(SF3,J) → MAR, L*(X ∧ Y) → A, A → M(MAR), h28 → h29
h29	0 → RW, A → M(MAR), h29 → hC

<u>TST</u> if $M(F1)_{0-L1} \geq M(F2)_{0-L1}$, then 1 → TAG, else 0 → TAG

h1	SL1 → I, 1 → Y, 1 → Z, 1 → R, h1 → h2
h2	(SF2,I) → MAR, h2 → h3
h3	I-1 → I, (SF1,I) → MAR, M(MAR) → L, h3 → h4
h4	(SF2,I) → MAR, M(MAR) → X, (I ≥ 0)h4 → h5, (I < 0)h4 → h6
h5	I-1 → I, (SF1,I) → MAR, M(MAR) → L, L*(X ∧ Y) → A, r → R, h5 → h4
h6	L*(X ∧ Y) → TAG, h6 → hC

5. Sample application programs.

Radar data can be organized as a time series of column vectors; each vector component carries the echo information relative to a discrete range interval along the sector illuminated by the antenna beam. The collection of column vectors relative to a given part of the area covered by the radar ssystem is stored in the PE memories, the n-th components being assigned to n-th PE for processing. A new vector enters the PE memories at constant time intervals ...t_{j-2}, t_{j-1}, t_j... through the Parallel Input (Fig. 1), one component per PE. All components have to undergo the same processing, which produces an output vector containing the target information extracted from the original vector. The radar algorithms for the n-th component of a vector generally require the values either of neighbouring components of the same vector, or of the n-th components of a number of preceding vectors (in radar jargon, azimuth filtering and range filtering, respectively). The n-th components of consecutive vectors are stored in consecutive memory cells of the n-th PE; then the working area, that is the memory space containing the last input datum and the previous partially computed results, moves ahead in memory. The base register R0 is intended to keep track of the actual position of working area, so that programs can be written as if data were in a fixed position in memory.

As an example of program, we can show the program implementing the Moving Target Indicator (MTI) double canceller, that is a filter whose output component y(n,j) is the weighted sum of the input components x(n,j-2), x(n,j-1), x(n,j), with weights 1, -2, 1. The resulting formula is:

$$y(n,j) = x(n,j-2) - 2\ x(n,j-1) + x(n,j)$$

Usually the MTI filtering is used only for some vector components. The PE's which will actually execute the algorithm must have their TAG register set to 1; the T field in the instruction is therefore set to 0. We assume that the complex component x(n,j) is stored in the memory of the n-th PE, the real part in the memory word of address 0, the imaginary part in the memory word of address 1. The complex components relative to the two preceding vectors are stored in the following words: x(n,j-1) in the words having addresses 126 and 127; x(n,j-2) in the words having addresses 124 and 125. To adjust the position of the working area in the memory, it is then necessary to give an increment of 2 to the base register R0 at the end of each time interval t_j. We assume that the initial contents of R0 is zero. The result of MTI filtering is stored in the memory words having addresses 124 and 125. The program (Fig. 5) is constituted only by second class instructions, that is by instructions executed by the Slave Computer. For each instruction, the various fields are indicated one after another, separated by commas (the field b represents a blank).

```
1)  SHL, 0, 0, 126, 1,   b, 0, 122
2)  SHL, 0, 0, 127, 1,   b, 0, 123
3)   SB, 0, 0, 124, 0, 122, 0, 124
4)   SB, 0, 0, 125, 0, 123, 0, 125
5)   AD, 0, 0, 124, 0,   0, 0, 124
6)   AD, 0, 0, 125, 0,   1, 0, 125
```

Fig. 5. A program for the MTI double canceller

As a second example of program, we show the program implementing the Matched Filter for Pulse Compression (for a 13 element Barker code), that is a filter whose output component y(n,j) is given by

$$y(n,j) = \sum_{s=-6}^{+6} W_s \cdot x(n-s,j)$$

where the values of the weights W_s are:

$$W_{-6}=W_{-5}=W_{-4}=W_{-3}=W_{-2}=W_1=W_2=W_4=W_6= -1$$

$$W_{-1}=W_0=W_3=W_5=1$$

Usually, this filtering operation is executed by all

```
01)   ADD1, 1, 0, 124, 0, 124, 0, 122
02)   ADD1, 1, 0, 125, 0, 125, 0, 123
03)  MSBD3, 1, 0, 124, 0, 122, 0, 120
04)  MSBD3, 1, 0, 125, 0, 123, 0, 121
05)   SBU1, 1, 0, 122, 0, 120, 0, 122
06)   SBU1, 1, 0, 123, 0, 121, 0, 123
07)   SBD1, 1, 0, 122, 0, 120, 0, 122
08)   SBD1, 1, 0, 123, 0, 121, 0, 123
09)  MSBD3, 1, 0, 120, 0, 122, 0, 120
10)  MSBD3, 1, 0, 121, 0, 123, 0, 121
11)   SBU1, 1, 0, 122, 0, 120, 0, 122
12)   SBU1, 1, 0, 123, 0, 121, 0, 123
13)   SBU1, 1, 0, 122, 0, 124, 0, 122
14)   SBU1, 1, 0, 123, 0, 125, 0, 123
15)  MADU3, 1, 0, 124, 0, 122, 0, 120
16)  MADU3, 1, 0, 125, 0, 123, 0, 121
17)   SBD1, 1, 0, 122, 0, 120, 0, 122
18)   SBD1, 1, 0, 123, 0, 121, 0, 123
19)   SBU1, 1, 0, 122, 0, 120, 0, 122
20)   SBU1, 1, 0, 123, 0, 121, 0, 123
21)  MSBU3, 1, 0, 120, 0, 122, 0, 120
22)  MSBU3, 1, 0, 121, 0, 123, 0, 121
23)   ADD1, 1, 0, 122, 0, 120, 0, 122
24)   ADD1, 1, 0, 123, 0, 121, 0, 123
```

Fig. 6. A program for a Matched Filter for Pulse Compression

PE's, then the T field is set to 1. The complex inputs to the filter are stored in the memories of $PE_{n-6}, \ldots, PE_n, \ldots, PE_{n+6}$ in the words having addresses 124 and 125. The complex output is stored at the same addresses in the memory of PE_n. We assume that the initial contents of RO is zero. The program is shown in Fig. 6.

6. Conclusions

The Processing Element of SMA has been presented, which is designed to be easily integrated on a custom chip to obtain a high system compactness. Microprogramming and programming examples have been reported. More complete informations on system structure, instruction set, microprograms and performance evaluation are reported in ref[12-14], but cannot be completely published, owing to security classification. Further study is in progress to enhance the capabilities of the Processing Element up to the limits imposed by the present technology, by enlarging to four-bits bytes the serial arithmetic.

References

1 P. Corsini, G. Frosini, G. Galati, F. Grandoni, M. La Manna, Serial Microprocessor Array for Radar Signal Processing, EUROMICRO, 1976, North-Holland Publishing Company, pp. 317-325.

2 J.A. Githens, An Associative Highly-Parallel Computer for Radar Data Processing, Parallel Processors, Systems, Technologies and Applications, Edit by Hobbs etc., Spartan Books, 1970.

3 B.A. Crane, M.J. Gilmartin, J.H. Huttanhorff, P.T. Rux, R.R. Shively, PEPE Computer Architecture, COMPCON, 1972, pp. 57-60.

4 G.D. Bergland, C.F. Hunnicutt, Application of PEPE to Radar Data Processing, COMPCON, 1972, pp. 65-68.

5 K.J. Thurber, An Associative Processor for Air Traffic Control, AFIPS Spring Joint Computer Conference, 1971, pp. 49-59

6 J.A. Rudolph, A Production Implementation of an Associative Array-Processor-STARAN, AFIPS Fall Joint Computer Conference, 1972, pp. 229-241.

7 K.E. Batcher, STARAN Parallel Processor System Hardware, AFIPS National Computer Conference, 1974, pp. 405-410.

8 B.P. Shay, Design Considerations of a Programmable Predetection Digital Signal Processor for Radar Application, Information System Group, Naval Research Laboratory, NRL Report 7455, December 1972.

9 H. Hebert, Parallel Processing of Radar Information by Associative Processors, Proceedings of IEE International Conference on Radar: Present and Future, October 1973, pp. 302-310.

10 G.R. Couranz, M.S. Gerhardt, C.J. Young, Programmable Radar Signal Processing Using the RAP, Sagamore Computer Conference on Parallel Processing, Springer-Verlag, Berlin, 20-24 August, 1974; pp. 37-52.

11 C.E. Muheua, P.G. McHuth, W.H. Drury, B.G. Laird, The Parallel Microprogrammed Processor (PMP), Proceedings of 1977 Radar Conference, London, October 1977, pp. 97-104.

12 P. Corsini, G. Frosini, G. Galati, F. Grandoni, M. La Manna, Elaboratore Multifunzionale per Segnali Radar - Parte Prima, Rapporto 76002/P, Convenzione Selenia-CNR, February 1976.

13 P. Corsini, G. Frosini, G. Galati, F. Grandoni, M. La Manna, Elaboratore Multifunzionale per Segnali Radar - Parte Seconda, Rapporto 76008/P, Convenzione Selenia-CNR, September 1976.

14 P. Corsini, G. Frosini, G. Galati, F. Grandoni, M. La Manna, Elaboratore Multifunzionale per Segnali Radar - Parte Terza, Rapporto 77001/P, Convenzione Selenia-CNR, January 1977.