ARCHITECTURE DEVELOPMENT FOR A GENERAL PURPOSE DIGITAL FILTER

R.J. Karwoski

1. INTRODUCTION

Advances in solid state technology have created new awareness in the rapidly growing field of digital signal processing. Early ad hoc approximations to analog processing techniques have given way to modern sophisticated digital methods.

General purpose computing machines are capable of performing a wide variety of mathematical signal processing algorithms for filtering and spectral analysis, but fall short in the areas of speed and programming ease. As a result, interest is shown in the dedicated processor for performing specialized processing tasks. Often configured as peripherals, these machines are linked to the general purpose host machine whose function is reduced to controlling such devices.

A number of approaches exist to implement a signal processor. For nonrealtime processing, large computers with Fortran or other high level scientific language capability are commonly employed. The data to be processed is permanent on some storage medium like tape, and processing speed is generally not a critical issue in this case. This paper is concerned mainly with realtime signal processing, particularly in the area of digital filtering. The following sections describe the hardware for a flexible fast digital filter computing machine which can be programmed easily.

2. BACKGROUND

Maintaining reasonable fidelity in a signal imposes a minimum sampling rate constraint on the processor. This is a relatively inflexible quantity and often allows limited time between data samples to perform the necessary steps to implement the desired filter. Computation efficiency is therefore a main concern. The proposed machine maximizes computing efficiency in three ways:

1) Separation of control and data functions

2) Optimization of data arithmetic unit (DAU)

3) Isolation of all control functions.

Separation of control and data functions allows data computations and instruction setup to be done simultaneously, thereby helping to maximize the execution speed for a given computer instruction. The data arithmetic unit (DAU) which performs the necessary arithmetic on the data to implement the filter is a specialized structure with optimum properties for digital filter work. It is under the unconditioned control of the main controller and is not available for calculating effective addresses or doing any other control oriented computations.

In the implementation of a digital filter, most instructions are accompanied by a multiplier coefficient, memory delay information, and a general purpose control word. In the described machine, these parameters are stored in four separate memories allocated specifically for them. Therefore, they can be fetched simultaneously and instantaneously directed to their proper destinations via a small amount of independent control logic. A general purpose microprocessor would normally process these items sequentially through the central processing unit (CPU) thereby consuming a great deal of time. In addition, the programming details with a general purpose machine are unnecessarily complicated. The parallel processing properties of the new machine make programming very simple because each item is independently specified and processed by individual control circuitry.

Figure 1 shows the basic machine concepts in block diagram form.

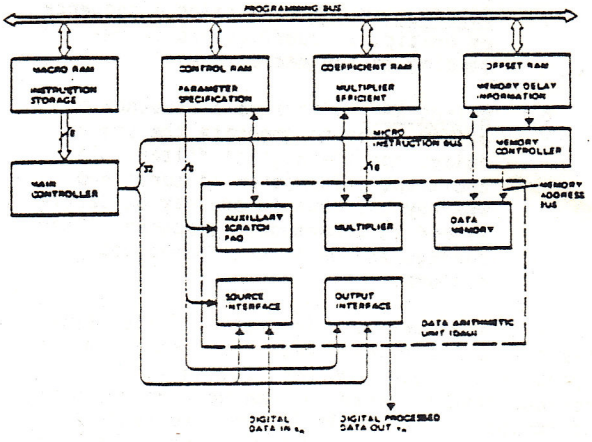
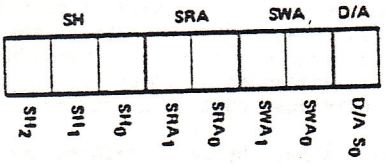


Figure 1. Fast Processor Block Diagram

The program bus originates from an outside source such as a microcomputer which may have a software interpreter to load each memory according to the filter program. The function of each memory word is described below:

a) Macro RAM: Contains the macro instructions which define the DAU operations. The instructions are stored sequentially in memory just as they are to be executed by the machine.

b) Control RAM: Contains the digital words which define certain machine parameters that are effected during execution of the corresponding macro instructions. The value of the word uniquely defines which scratch pad location is being written and/or read. When there are multiple output ports (e.g., four D/A converters) or multiple input sources, this information can be included in the word. Figure 2 shows the format for the 8 bit parameter word used.



D/A: D/A select (0-1, 1 Bit)
SWA: Scratch pad write address (0-3, 2 Bits)
SRA: Scratch pad read address (0-3, 2 Bits)
SH: Shift code (0-8 places, 3 Bits)

Figure 2. Parameter Control Word

c) Coefficient RAM: Contains a sequence of multiplier coefficients corresponding to the macro codes.

d) Offset RAM: Contains a sequence of offset words to generate the proper delays for the digital filter. In addition, information is contained whereby the data memory may be partitioned to accommodate cascaded filter configurations or timeshared applications.

A common operation in digital filter synthesis involves reading data from memory, multiplying it by another number, adding it to a previous result stored in some working register, and restoring the result in the same register. This instruction is the MEM MAC instruction, so named because it involves a memory read, multiplication, and accumulation. It will be shown in the following section that this operation and similar ones (which will comprise the machine macro instruction set) will take a single microcycle to perform.

The DAU is informed of its data routing plan by the instruction sequencer (in the main controller) during one clock cycle. The sequencer transmits the proper enabling signals to the correct DAU elements via the microinstruction bus. The data, multiplier coefficient, main memory address, and microinstruction all arrive at their destinations simultaneously. By the next clock edge the data has been piped through the appropriate DAU elements and is settled into its final resting place (an accumulator register for the MEM MAC instruction). During the next clock cycle, a new coefficient accompanied by a memory address word and prescribed parameter word (if necessary) arrive along with the microcode. The data during this interval are piped in the appropriate directions and stored again.

The 24 bit microinstruction is usually a direct translation of the 8 bit macro code stored in the macro ram. However, canned filter routines may be stored as a sequence of microcodes. In this case the macro instruction is used to trigger the sequence which is internally generated. This technique is basic to modern microcoded machines and is described later.

3. DAU ARCHITECTURE

Confronted with the myriad of possible digital filter configurations, the designer's first compulsion is to create a completely universal DAU bus structure as shown in Figure 3. Each DAU element has direct and independent

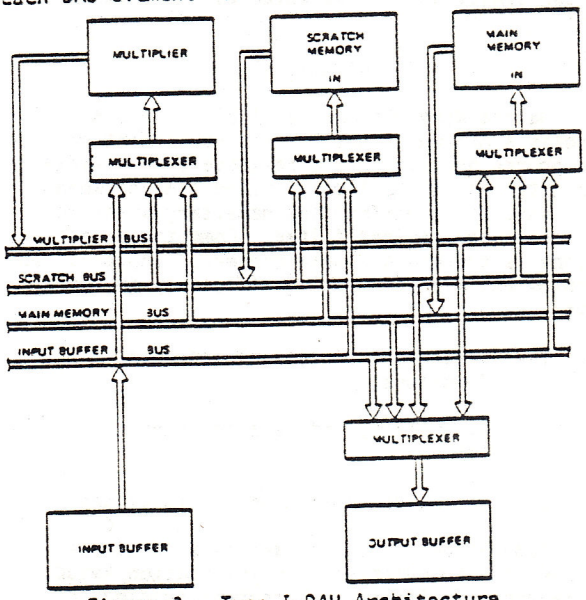


Figure 3. Type I-DAU Architecture

access to the output of any other DAU element. This allows data to run simultaneously on any number of paths, minimizing sequential steps and computer time, while allowing the synthesis of all filter forms.

The flexibility afforded in the Figure 3 configuration is not necessary, as efficient execution of even the most complicated algorithms involves variations of relatively few routines with the TDC1010J multiplier-accumulator. These routines can be designed to require a maximum of two data buses running at any given execution time with little or no compromise in speed. Arriving at the exact structure for a given dedicated processor may require tracing out data paths on the architecture for some typical program executions. After a number of these the design can eliminate unused paths, minimize his architecture, and arrive at a suitable micro-instruction set.

Figure 4 illustrates an DAU section which provides very efficient execution of all digital filter configurations at modest hardware cost. Data is channeled in and out of the DAU via the input and output storage elements (X and OR). These may be A/D and D/A interfaces or additional memory. The heart of the DAU is the TCD1010J multiplier-accumulator, and most of the data computations are done there. Local rapid access storage is handled in the scratch pad memory (Sc), and the bulk data storage is done in the main memory which provides the necessary data delays for digital filter implementation. An extended range adder is linked to the scratch memory so that accumulations can be performed for parallel configurations. The extra accumulator frees the 1010 from the responsibility of having to perform simple group addition tasks which may be done automatically as multiplier products are shipped off for local storage. A combinatorial shifter is provided at its output so that the accumulated data may be rendered instantaneously compatible with the main 16 bit bus. The shifter output is linked to the main bus while the actual scratch pad memory output drives the auxiliary bus into the multiplier input multiplexer (MUX).

As an introduction to the efficiency afforded by the auxiliary bus the following example is given: a second order canonical filter is to be implemented. The signal flow graph is shown in Figure 5.

Input x_n is added to past (extracted from memory) data products $-b_0 x_1$ and $-b_1 x_2$. The result is written back into memory as indicated (node 3). During this write cycle the MAC input

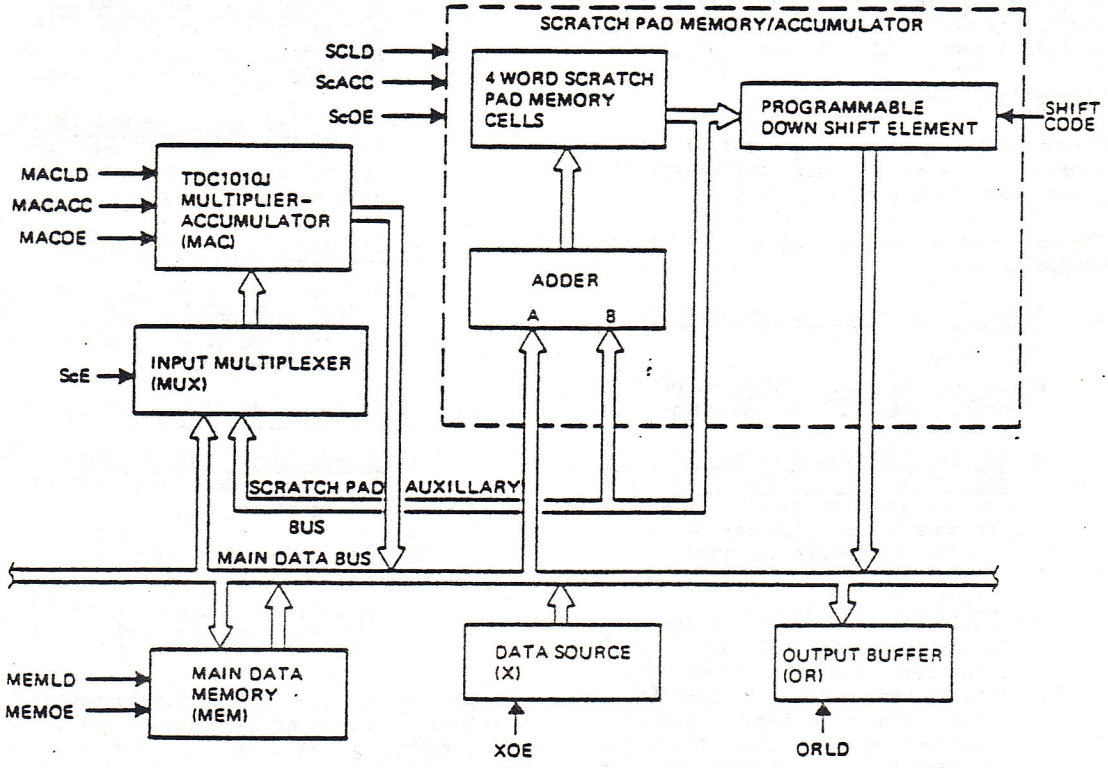


Figure 4. Type 2 Architecture

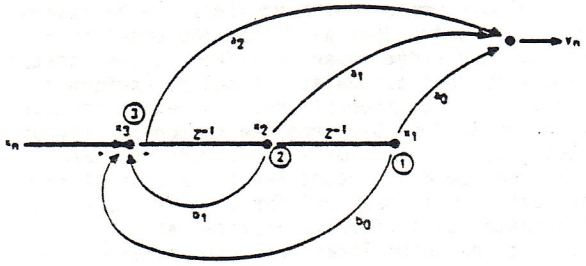


Figure 5. Signal Flow Graph

section is free to do another multiply or multiply-accumulate. Unfortunately, during a write cycle the memory I/O bus is tied up and cannot provide the necessary data (x_1, x_2 or x_3) for the multiplications to form output y_n. If during the recursive operation x_1 (or x_2) is simultaneously stored in Sc while it is being transferred to the MAC, it can then be used during the write cycle when the MAC would otherwise be idle. Data will be running on both buses during this time, resulting in the elimination of one microcycle for implementation of the filter. This constitutes a substantial savings in machine time, because four to six cycles are necessary to perform the complete algorithm (see instructions 3 and 5, Table 1, Section 3.2).

3.1 Controlling the DAU

Directing the traffic in and out of each DAU element is a relatively straightforward task and is performed by the main controller.

The main control signals at each DAU element are defined below:

a) Scratch Pad Memory/Accumulator (Sc) Controls

- Scratch Pad Load (ScLD). Clock enable used to load the memory.
- Scratch Pad Read Address, Scratch Pad Write Address (SRA, SWA). Memory address directs data in and out of memory; addresses may be tied together or remain separate.
- Scratch Accumulate (ScACC). Logic level activates scratch adder to perform accumulation. A logic 1 will allow data on the bus to be added to data in memory location specified by SRA and stored in memory location specified by SWA. When this signal is inactive a simple data write or read can be performed.
- Scratch output Enable (ScOE). Logic level enable shifter data on the main bus.
- Shift Code (S). n bit code specifies the number of places the scratch pad word is to be shifted down; hence, division by 2^n is possible.

b) Main Memory Controls

- Memory Output Enable (MEMOE). Logic level enables memory data onto the main bus.
- Memory Load (MEMLD). Clock enable used to load data into memory.
- Memory Address (MEMA). N bit code specifies the location of the memory word which is accessed.

c) TDC1010J Multiplier-Accumulator (MAC) Controls

- Multiplier-Accumulator Load (MACLD). Clock enable for loading data into X, Y, and accumulator registers.
- Multiply/Accumulate (MACACC). Active logic level directs an accumulate operation within the TDC1010J after multiplication.
- Multiplier Output Enable (MACOE). Active logic level allows the multiplier product onto the main data bus.

d) Multiplexer Input (MUX) Controls

- Scratch Pad Enable (ScE). Active logic level enables data into the MAC from the scratch memory via the auxiliary bus.

e) Input Source (X) Controls

- Source Output Enable (XOE). Active logic level enables X data onto the main bus.

f) Output Buffer (OR) Controls

- Output Load (ORLD). Clock enable for loading this element (register on buffer memory).

Data may be sent on the bus connecting one DAU element to another by enabling the output (e.g., MACOE) of the sender and the input clock enable for the receiver (e.g., ScLD).

All control lines are assumed to be a logic 1 for an enabling condition. This simplifies matters in the preliminary stages of the design. It is understood that each ALU element may have its particular requirements for loading and enabling. For example, the TDC1010J requires three input clocks and has five output enables. Also most logic will be tristate, requiring logic zero for an output enabling condition. These particulars may be taken care of with a small amount of random logic in the ALU control interface. The controller outputs during any given microcycle report the state of the machine and may be used as enabling signals when necessary, as well as the actual control signals.

Figure 6 is a simplified block diagram of the TDC1010J multiplier-accumulator (MAC), and Figure 7 is a timing diagram illustrating a loading and output enabling sequence of the 1010 during a typical computer cycle. Machine cycle T_i contains an instruction requiring a multiplication or multiply accumulate and that the previous product which is stored in the product register be transferred to the main bus. The 1010 contains input registers for X and Y operands. It is not necessary to waste a complete clock cycle to load X and Y registers since typical multiply-accumulate time for the MAC is under 150 nsec, and loading is not useful for anything other than multiplier operations. The clock interval may be subdivided such that loading, multiplying, adding, and final storage are carried out in various portions of a single microcycle.

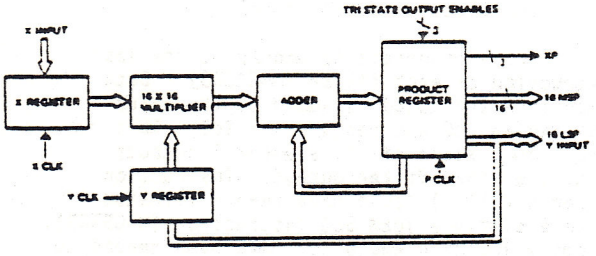


Figure 6. Simplified TDC1010J Multiplier-Accumulator Block Diagram

Due to the large number of inputs and outputs required for 16 x 16 double precision multiplication, it is necessary for some functions to share common pins on the TDC1010J. This is a slight problem for implementations requiring multiplication by numbers greater than 1. For the synthesis of second order filters, multiplication by 2 may be necessary. The required data shift by 1 bit (and subsequent

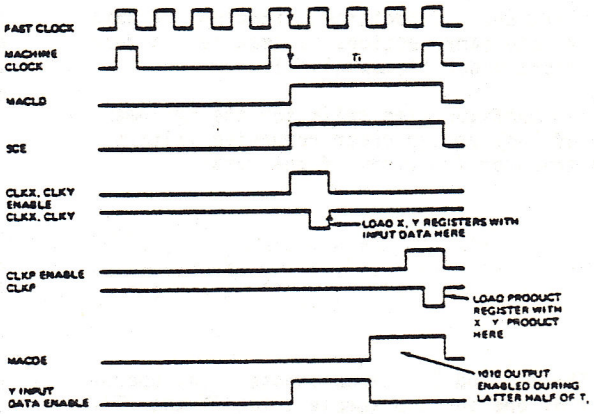


Figure 7. Typical TDC1010J Loading Sequence Timing Diagram

multiplication by 2) is effected by connecting P_{30} (second product MSB) to the MSB of the data bus (Figure 8). Unfortunately, Y_{15} (coefficient MSB) shares a pin with P_{15}, the new data bus LSB. These cannot be enable simultaneously, so the microcycle is divided so that the output data from the P register is on the bus during the latter half of the cycle; X and Y are loaded during the first half.

The additional combination logic necessary to accommodate the particulars depicted by the timing diagram results in a savings of microprom and overall machine program complexity.

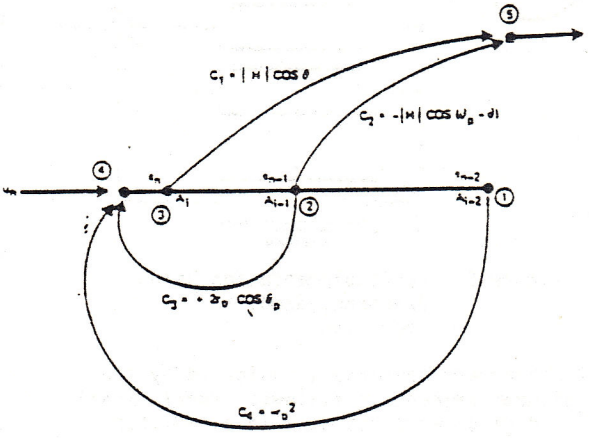


Figure 8. Prescribed Second Order Filter Signal Flow Diagram

3.2 Frequency Sampling Filter Synthesis Using Type 2 Architecture

This is probably the hardest filter to implement because it involves many types of instructions and the use of all of the ALU's

capabilities. In addition, "fencing" of data memory into many sections is required to form the second order elemental filters.

The configuration calls for the implementation of many second order recursive filters with transfer functions of the form:

$$H(Z) = \frac{|H(\theta_p)|\ [\cos\theta - \cos(\theta_p - \theta)\ Z^{-1}]}{1 - 2\ r_p \cos\theta_p\ Z^{-1} + r_p^2\ Z^{-2}}$$

The Z^{-1} and Z^{-2} terms denote delay operations of one and two sample times (T and 2T). As sighted in Section 2, these delays are usually implemented with RAM but may be envisioned as taps at T seconds and 2T seconds on a delay line.

Each filter output is summed to form the composite filter output (see Figure 9). Filter parameters $H(\theta_p)$, $\cos\theta$, $\cos(\theta_p - \theta)$, and $2r_p \cos\theta_p$ can be combined; and the signal flow graph for the transfer function H(Z) is shown in Figure 8.

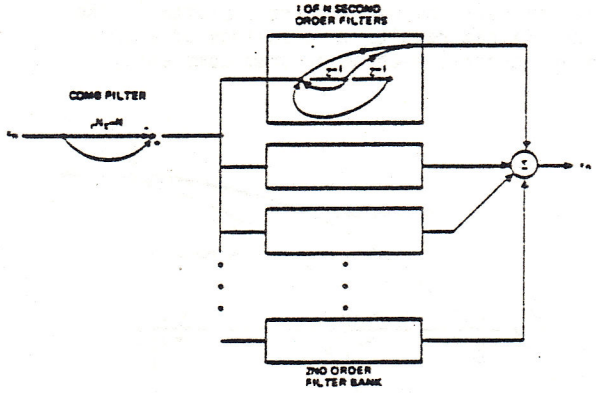


Figure 9. FIR Implementation Using Frequency Sampling Technique

The hardware operations indicated by the flow diagram proceed as follows: input signal u_n is a digital word residing in a register (scratch pad, A/D register, etc.) at a time nT. It is summed with words C_3e_{n-1} and C_4e_{n-2} to form $e_n = C_0U_n + C_3e_{n-1} + C_4e_{n-2}$; e_n is then written into memory at effective address A_i (node 3, the input of the delay line). e_{n-1} and e_{n-2} are the results of identical operations having occurred one and two data sample times before. They reside in memory at locations A_{i-1} and A_{i-2}, during sample time period nT. During the next sample time, (n+1)T, when x_{n+1} is to be processed, e_n will advance to node 2 (address A_{i-1}), e_{n-1} to node 1 (A_{i-2}), and e_{n+1} will be generated and written into memory at node 3 (A_i). The filter output is formed in a fashion similar to that just described for the feedback operation. C_1 and C_2 are the multiplying coefficients and $y_n = C_1e_n + C_2e_{n-1}$.

Including the input scaling multiplication (C_0U_n), five multiplications and four additions are necessary to implement this filter. If we assume a 150 nsec machine cycle time and can achieve one cycle per multiply/accumulate (multiply/add), the elemental filter computation will take 750 nsec. Usefulness of the scratch accumulation capability is realized for this computation sequence. The filter outputs can be accumulated as each is piped from the 1010 output register through the adder into a scratch pad location. Two machine cycles per filter are eliminated. Ignoring initial setup and the small amount of overhead time which is likely to occur, a 100 frequency sample filter (i.e., a 200th order filter) will take 75 μsec to implement.

Table 1 lists a feasible instruction sequence for the frequency sampling filter. The active data paths are indicated in the group of diagrams in figure 10. Figure 11 is a detailed timing diagram for the sequence.

The comb filter which precedes the filter bank is implemented in instructions 2, 3, and 4. Instruction 1 contains a NOP but may be needed for program initialization. Instructions 6 through 11 implement the first second order section. Twelve through 17, 18 through 23, etc., are repeats of 6 through 11 with the exception that coefficients differ for all the filters. All the filter outputs are automatically accumulated (added) in Sc0.

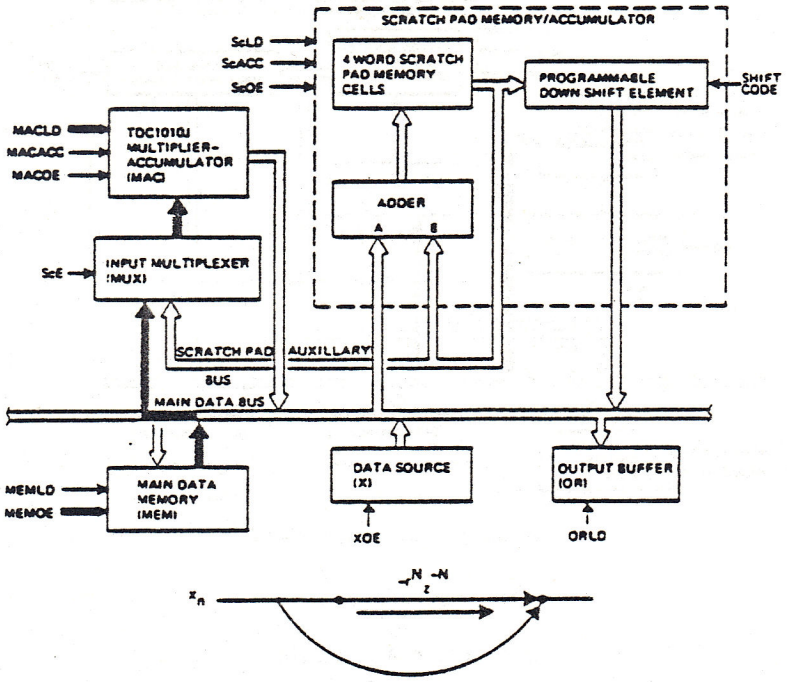
It is appropriate to emphasize the last instruction of each filter (MACACC0). Data taken from the 1010J is added to the value residing in Sc0 and restored in Sc0. This running accumulation of all elemental outputs forms the final filter output. Instruction number 5 (C1Sc1) could have been eliminated if 11 were simply a load Sc0 instruction (MACST00). An empty location would not have been needed to prepare for the first filter output data dump. However, from a programming standpoint, the complete sequence (steps 6 through 11) can be stored as a canned routine in the microprogram PROM, thereby simplifying the program. The programmer need only call up "second order filter," specify the coefficients, and forget about the details of the microinstruction sequence. Instead of six program steps for each filter, he would have to program only one each time he wanted to use the filter.

Table 1. Macro Instruction Sequence for Frequency Sampling Filter

Instruction Description	Instruction	Data Notation	Multiplier, C
1. Nothing (or program initialization)	NOP	--	--
2. Multiply MEM data by r^N; store in 1010 register	MemMult	C·Mem→MAC	$-r^N$
3. Multiply input x data by C_i, add result to value residing in 1010 register, restore in 1010 register	XWM/XMAC	X+MAC→MAC X→MEM	C_i (Comb input scaler)
4. Store 1010 output register value in Sc address 1	MACST01	MAC→Sc1	
END OF COMB COMPUTATIONS			
5. Clear scratch pad location zero	C1Sc	0→Sc0	--
6. Multiply data in Sc address 1 by C_0, place product in 1010 output register	Sc1Mult	C·Sc1→MAC	C_0 (canonical input scaler)
7. Multiply e_{n-1} by C_3. add result to value residing in 1010 output register, place this sum in 1010 output register; store e_{n-1} in Sc address 2	MemMAC/ST02	C·Mem+MAC→MAC MEM→Sc2	$C_3 = 2r_p \cos\theta_p$
8. Multiply e_{n-2} by C_4, add this product to value residing in the 1010 output register, place this sum in the 1010 output register	MemMAC	C·Mem+MAC→MAC	$C_4 = -r_p^2$
9. Load MAC data into MEM; multiply present data in Sc address 2 by C_2, store in 1010 output register	MWM/Sc2Mult	MAC→MEM C·Sc2→MAC	$C_2 = -\|H\| \cos(\theta_p - \theta)$
10. Multiply e_n by C_1, add it to data residing in 1010 register, restore result in 1010 register	MemMAC	C·Sc2+MAC→MAC	$C_1 = \|H\| \cos\theta$
11. Add data in 1010 register to data in Sc address 0 and restore the result in Sc address 0	MACACC0	MAC+Sc0→Sc0	--
FILTER NO. 1 COMPLETE			
12. Multiply data in Sc address 1 by C_0, place product in 1010 output register	Sc1Mult	C·Sc1→MAC	C_0 = input scaler

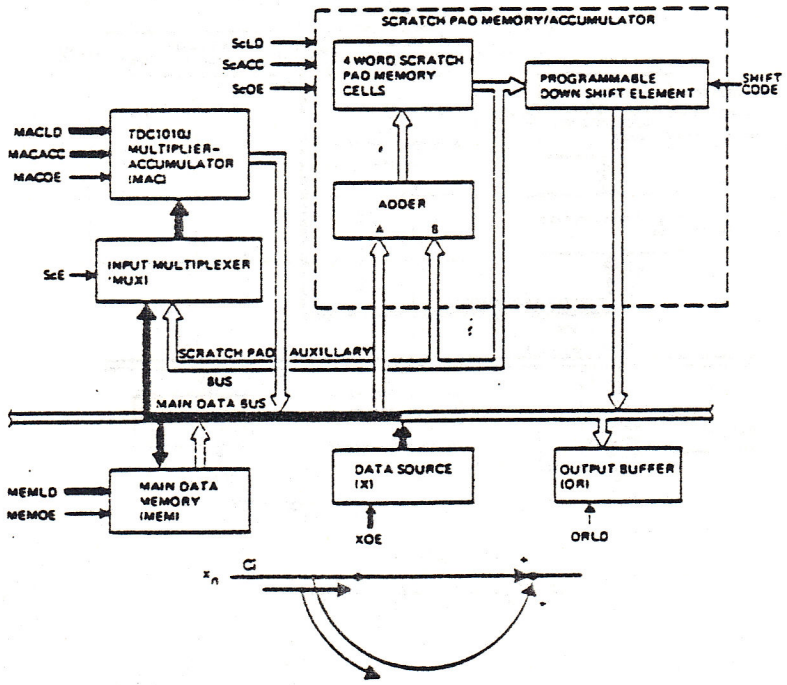
Table 1. Macro Instruction Sequence for Frequency Sampling Filter (Continued)

Instruction Description	Instruction	Data Notation	Multiplier, C
13. Multiply e_{n-1} by C_3, add result to value residing in 1010 output register; place this sum in 1010 output register; store e_{n-1} in Sc address 2	MemMAC/ST02	C·MEM+MAC MEM→Sc2	$C_3 = 2r_p \cos\theta_p$
14. Multiply e_{n-2} by C_4, add this product to value residing in the 1010 output register, place this sum in the 1010 output register	MemMAC	C·Mem+MAC→MAC	$C_4 = -r_p^2$
15. Load MAC data into MEM; multiply present data in Sc address 2 by C_2, store in 1010 output register	MWM/Sc2Mult	MAC→MEM C·Sc2→MAC	$C_2 = -\|H\| \cos(\theta_p - \theta)$
16. Multiply e_n by C_1, add it to data residing in 1010 register, restore result in 1010 register, restore result in 1010 register	MemMAC	C·Sc2+MAC→MAC	$C_1 = \|H\| \cos\theta$
17. Add data in 1010 register to data in Sc address 0 and restore the result in Sc address 0	MACACC0	MAC+Sc0→Sc0	--
FILTER NO. 2 COMPLETE			



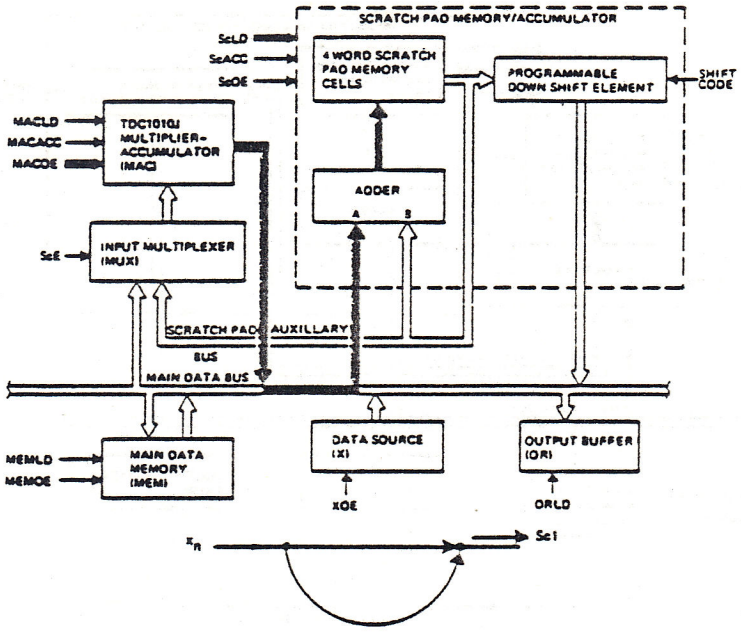
Instruction 2 – MemMult

Figure 10



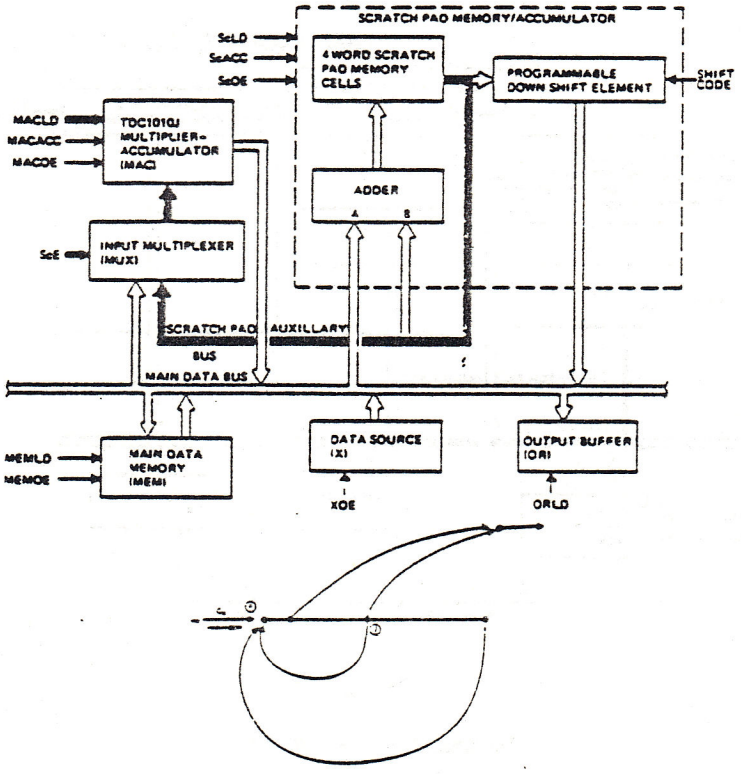
Instruction 3 – XWM/MAC

Figure 10



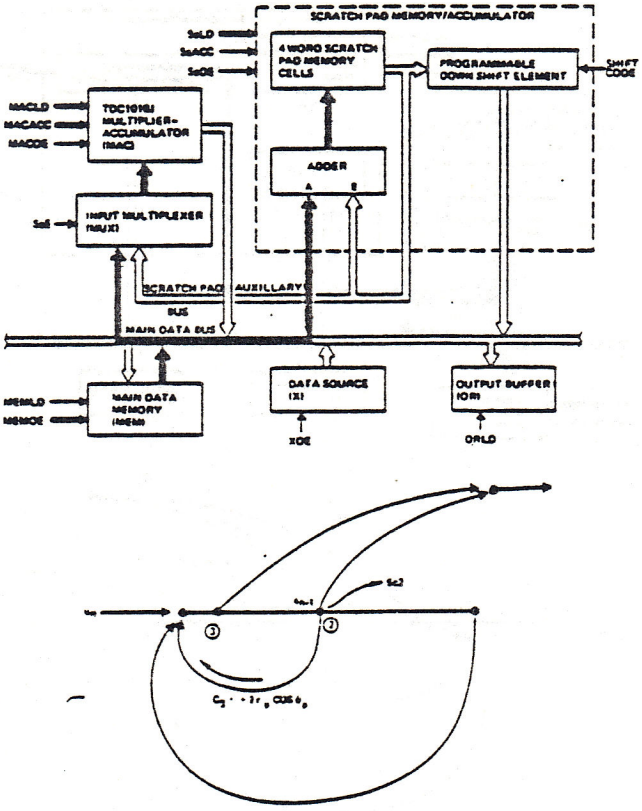
Instruction 4 – MACST01

Figure 10



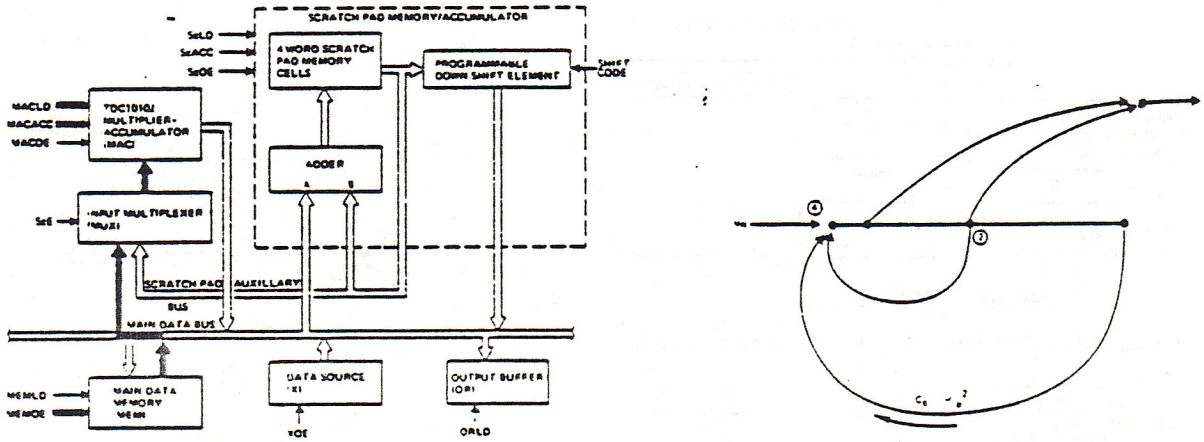
Instruction 6 – Scl Mult

Figure 10



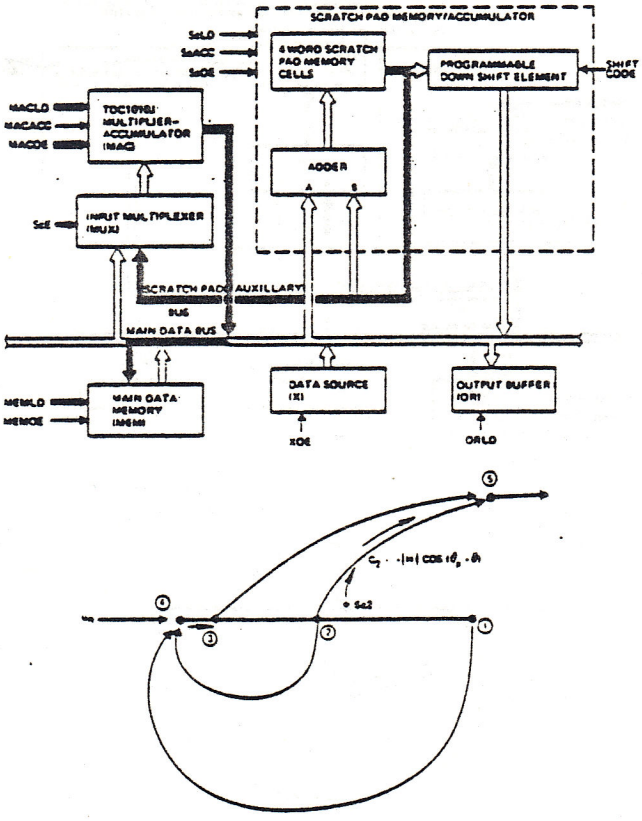
Instruction 7 – Mem MAC/STO2

Figure 10



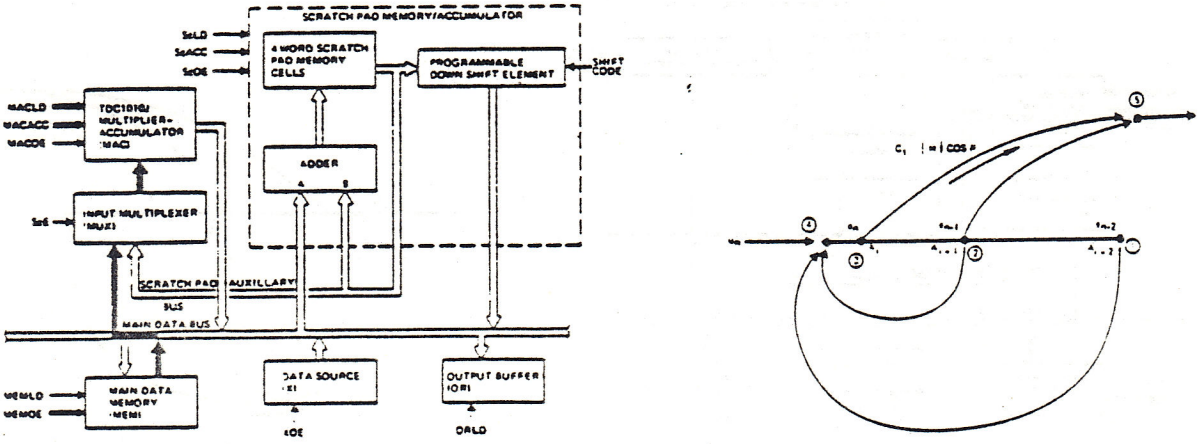
Instruction 8 – MemMAC

Figure 10



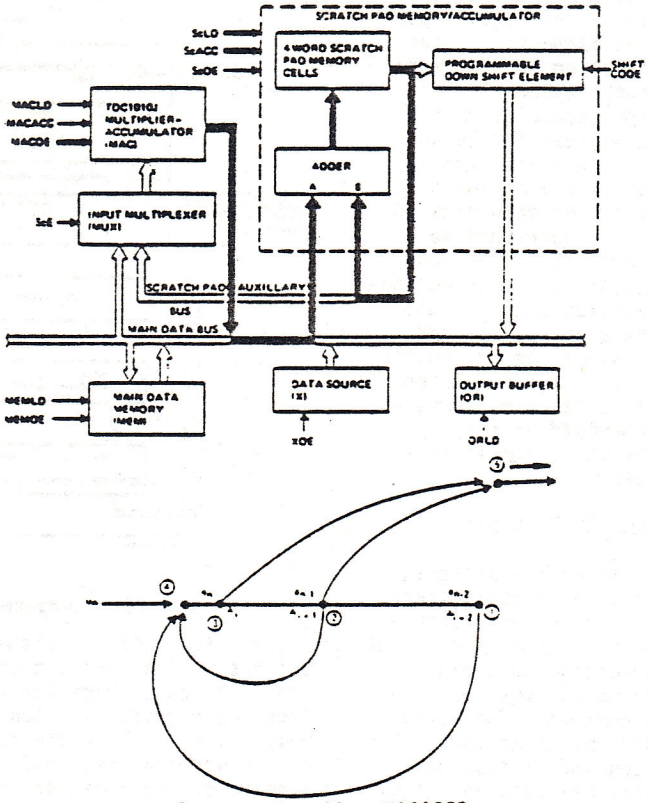
Instruction 9 – MWM/Sc2Mult

Figure 10



Instruction 10 – MemMAC

Figure 10



Instruction 11 — MACACCO
Figure 10

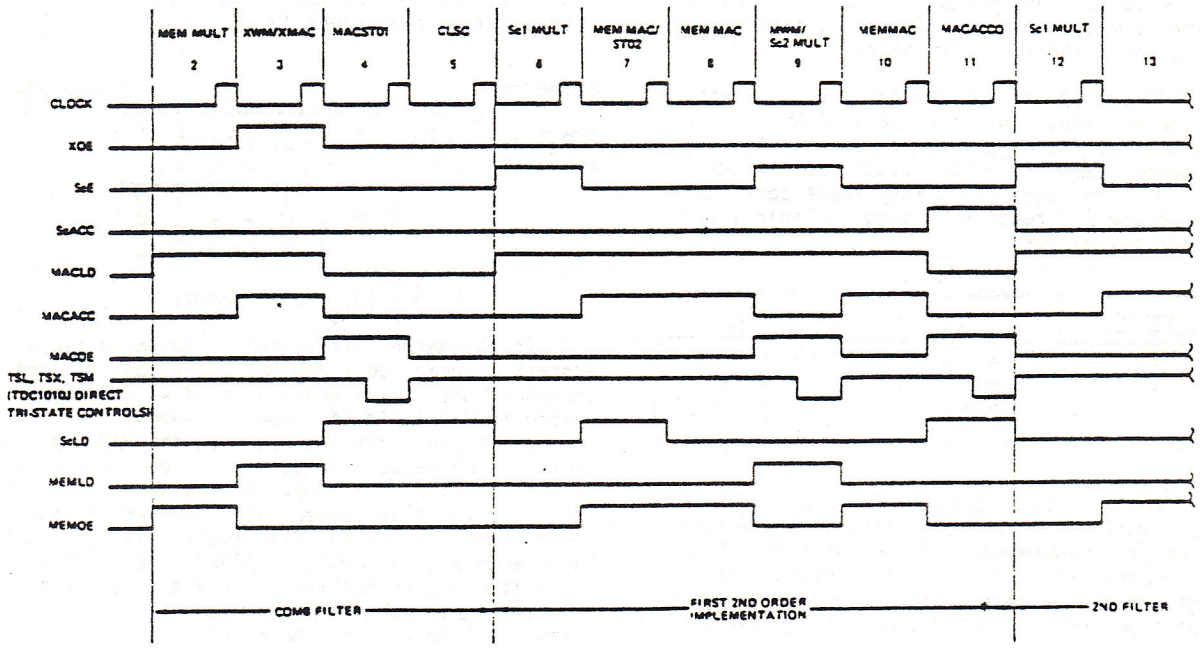


Figure 11. Timing Diagram for Frequency Sampling Filter

A point which should be addressed at this time deals with the shifter in the scratch pad memory. Accumulation of the elemental filter outputs in the frequency sampling structure may very well lead to word growth beyond 16 bits. This is not a problem in the accumulation instructions (i.e., MACACC0) because Sc0 is an extended range register. The problem lies in extraction of this data for user evaluation because it must be brought to the main data bus before it is shipped off. The nonrecursive coefficients (C_1 and C_2) could be scaled down in anticipation of the situation, but objectionable signal-to-noise degradation could result due to the attenuated signals on each filter output path. The shifter allows the arithmetic to be performed at its basic 16 bit precision. The result is rescaled only after the accumulations are completed, and roundoff errors may be substantially reduced. The shift number is included in the parameter word.

4. COMPUTER CONTROL UNIT (CCU)

All of the DAU's activities are directed by the computer control unit (main controller). Its primary function is to receive user programmed instructions, interpret them, and direct the DAU to perform the corresponding computations. For general microcomputer systems, controllers can become quite complex. The development of a flexible, yet uncomplicated controller which is particularly suited for processing realtime single and multichannel data for digital filtering is briefly discussed. Those familiar with microprocessor architecture will recognize its fundamental structure as that for a simple microcomputer. It can be expanded to any desired level of complexity.

Figure 12 defines the total control aspect of the machine. The user loads MACRO RAM with the desired filter program sequence. For the frequency sampling filter described in Section 3.2, hypothetical binary codes corresponding to the instructions listed in Table 1 are loaded as follows:

Macro RAM Address (Hex)	Macro Instruction	Macro Binary Code (Hypothetical)
0 0	NOP	0 0 0 0 0 0 0 0
0 1	MemMult	0 0 1 0 0 0 0 0
0 2	XMM/XMAC	0 0 0 1 0 1 0 0
0 3	MAC ST01	0 0 1 0 0 1 0 1

Macro program codes are read sequentially from memory under the direction of a RAM counter which is incremented by the controller. Each instruction is stored into the instruction register on the next clock following its extraction from memory. At this point, the macro

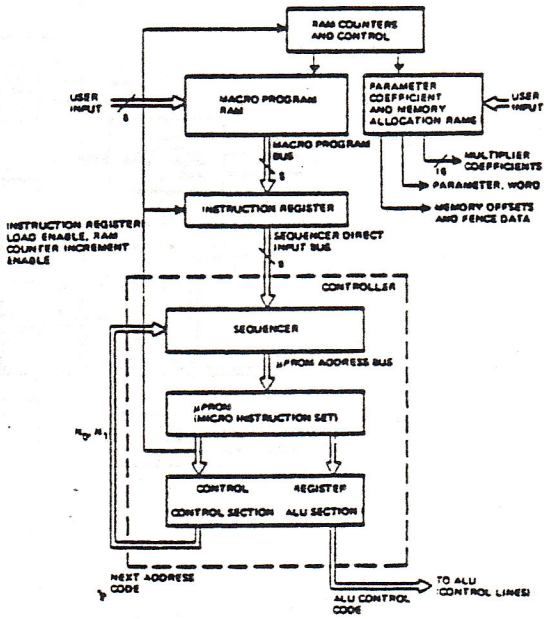


Figure 12. Computer Control Unit

instruction code is on the sequencer direct input bus. (The instruction to follow sits at the instruction register input awaiting entry into the register.) When the sequencer is not busy, it will allow the code to pass through it to the uprom address bus. This code is the address of the location in prom which contains the desired DAU microcode. For example, locations A_i and A_{i+1} in uprom may contain microinstructions MEM and MAC and MEM MULT, respectively (see Figure 13).

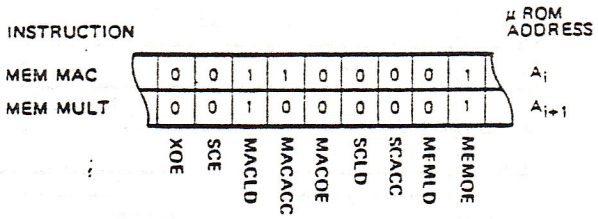


Figure 13. μprom Format

If the macro instruction residing in macro address 04 required a MEM MAC operation for some sequence, the binary equivalent of A_i would be placed in MACRO RAM 04. When A_i eventually arrived on the uprom address bus, the DAU control code corresponding to MemMAC would appear at the output of the prom. On the next clock this code would be loaded into the control register whose outputs would activate the required DAU elements (i.e., MACLD, MACACC, and MEMOE). The next microinstruction to be executed would be present at the uprom output awaiting its entry into the control register.

The sequencer's true capabilities are realized when large filers with many identical sections are to be implemented, or when individual filters are to be timeshared. The frequency sampling the filter is a perfect example since each second order canonical filter in the filter bank is identical in form. The microcode for instructions 6 through 11 can be stored in the microprom in the exact sequence that they appear in Table 1. The filter can be executed by placing the prom address which contains the first microinstruction (Scl Mult) into the desired macro RAM location. This uprom address is termed the "starting address". After it is extracted from memory and then stored in the instruction register, the sequencer gates it through to the uprom address bus as previously described. The microprom outputs the correct microcode for SC1Mult which is clocked into the control register. At this point, the sequencer does not take the next macro instruction which is still in the macro memory. Instead, it generates the next uprom addresses internally while the MACRO RAM counter and instruction register states remain frozen. If uprom addresses 80, 81, 82, 83, 84 and 85 contained the filter sequence, the user would program 80 into macro ram. The sequencer would recognize the control portion of the microcode associated with address 80 as the first of six microaddresses of a canned routine. It would then generate 81 through 85. At the end of the routine, with the microcode corresponding to address 85 in the control register, the sequencer would allow the next macro instruction (which would be residing in the instruction register) through to the uprom bus. The prom output at this time would represent the starting instruction for the next canned sequence or simply another single cycle macro instruction.

The timing diagram (Figure 14) for the Table 1 program gives a clear picture of the events described. The first five macro instructions occupy one machine clock interval each. Each takes two clock times to get from macro memory to control register output. There is a one cycle delay between macro memory and instruction register and another delay from there to the control register.

The sixth and seventh macro instructions, represented by decimal 80, are the starting addresses for the first two second order filters. Instruction 6 is clocked into the instruction register, and the macro counter is incremented to instruction 7 which appears at the instruction register input. Meanwhile, 6, which is on the direct bus to the sequencer, is gated

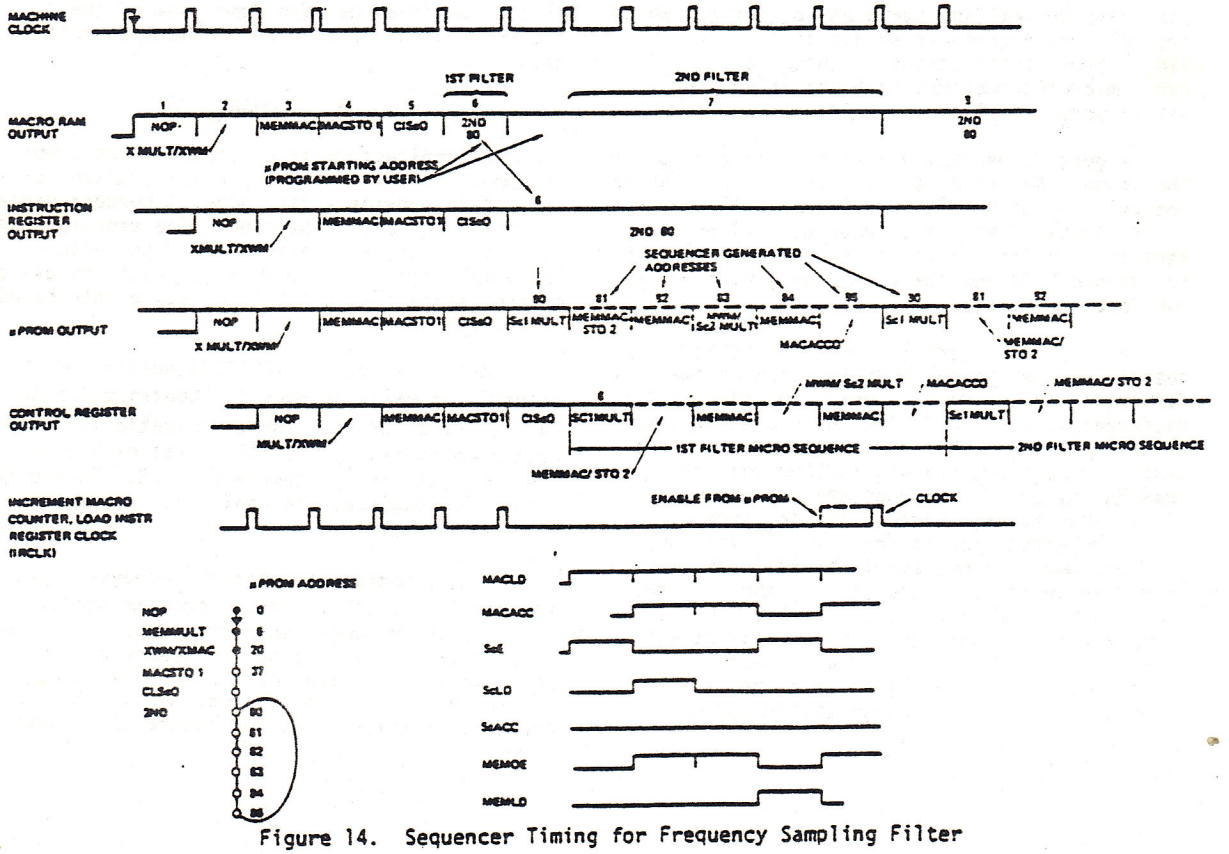


Figure 14. Sequencer Timing for Frequency Sampling Filter

through to the uprom address bus; 5 is in the control register being executed at this time.

On the next clock cycle the SCl Mult microinstruction jumps into the control register, but the clock to the macro counter and the instruction register (IRCLK) is inhibited. The instruction register retains the first filter starting address and the macro memory output still shows the second filter starting address.

IRCLK is inhibited for the next four cycles while the rest of the sequence is performed (i.e., MemMAC/STO2, MemMAC, MWM/Sc2Mult...). When the last instruction (MACACCO is on the uprom output, the clock is enabled. The macro counter increments to 8 and the instruction register receives instruction 7 as MACACCO is dumped into the control register. The control portion of the MACACCO microcode instructs the sequencer to gate in instruction 7 (micro prom address 80).

IRCLK is enabled by a uprom control signal which is activated when the last microinstruction of each routine is in uprom. This rule is perfectly general and pertains to routines which are only one microcycle in duration (i.e., single cycle instructions). Hence, all single cycle microinstructions have the IRCLK enable bit in uprom enabled.

Figure 15 emphasizes the control portion of the uprom. N_0 and N_1 are the control bits which decide in which mode the sequencer will operate. In its simplest mode the sequencer allows instruction register outputs through to the uprom. For canned routines the sequencer generates its own uprom addresses.

Given that a certain microinstruction resides in the control register (and is therefore being executed) the control portion of its microcode is used to direct the sequencer into a certain mode. If the microinstruction represents a single cycle macro instruction, then code N_0, N_1 should instruct the sequencer to allow a new address through to the uprom. If the microinstruction is the starting instruction or some intermediate instruction for a filter sequence, then code N_0, N_1 should direct the sequencer to continue generating successive uprom addresses. When the final microinstruction is in the control register the corresponding control code will once again gate the instruction register output through to the uprom.

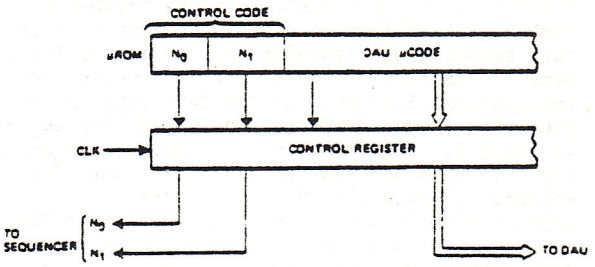


Figure 15. Microprom and Control Register Format

N_0, N_1 is therefore termed the "next address" code, since it indicates from where the uprom address for the next instruction will originate. Exiting from the filter sequence may be done using a third variation of N_0, N_1. A looping routine may be set up whereby a sequence of DAU no operation (NOP) instructions are performed while the sequencer awaits the arrival of a new data sample clock. This may be a requirement for systems with fixed sample rates and relatively short program length. The following section defines the main functions of the sequencer with respect to the "next address" code.

5. SUMMARY

An architecture is presented which combines arithmetic pipelining and parallel control processing to provide a fast general purpose digital filter computing machine. The processor can be built relatively inexpensively and with a reasonable amount of hardware through the use of present state of the art large scale integrated circuitry (LSI).

Specifically, the TDC1010J multiplier-accumulator makes an enormous contribution to the reduction of size and minimization of circuit complexity. It is noted that many areas of the processor lend themselves to LSI implementation. For example, the controller might be a good candidate.

An LSI package containing sequencer next address logic, and the micro program store would simplify the hardware significantly. The same holds true in the case of the memory controller and remaining portions of the DAU. With digital filtering becoming so popular, we look for advances in these areas for future LSI products.