FUNCTIONAL MEMORY TECHNIQUES APPLIED TO THE MICROPROGRAMMED CONTROL OF AN ASSOCIATIVE PROCESSOR

C.V.W. Armstrong

Dept. of Computer Science, University of Edinburgh

Edinburgh, Scotland

Summary - Consideration is given to the separation of the data and control structures of a microprogrammed processor. The use of functional memory techniques to provide a suitable medium for containing and processing the control structure and giving one possible solution to this separation problem is described. The design considered is that of an associative processor but the techniques involved are applicable to other types of processors. Some of the advantages of this approach are given, together with their implications in the light of advances in microprocessor technology and cellular logic.

Introduction

The Control Structure

The microprogrammed control unit of a processor has the task of fetching and executing instructions. These two functions can be separated out and handled by different control units. Similarly, the execution unit can be divided into a section that processes the data and a section that processes the control structure related to that data. Separation of this sort is useful because each control section is specialised for a particular type of operation and can work in parallel with the operation of the other sections. It seems possible to separate out these control sections[1] and in fact the use of a microprogrammed control unit can go some way towards making this separation. However, when the control structure requires processing, the microprogrammed control unit may borrow resources provided primarily for the processing of the data structure and parallel operation is no longer possible.

Take for example the case of the Interdata Model 70 and related models, a minicomputer where vertical microprogramming predominates. Testing may require the use of both S and B busses and does not allow multiway branches to take place. This, for example, would be very useful in the processing of interrupts. The processing of the control structure preempts any processing of the data structure during microroutine execution. In fact, the processing of the control structure is constrained by the data processing architecture of the processor[2]. A number of instructions have the same microroutines except for minor changes, which are restricted to small differences in the processing of the data structure. A means of separating out these two control sections may go some way in compacting the amount of space required to hold the differing microroutines into one where no such replications are necessary.

The attempt is made here to show how the use of functional memory techniques may allow a single microprogrammed control unit to process, in parallel, the data and control operations involved in executing instructions. This approach may be useful when a number of modules, such as microprocessors, must be controlled in parallel.

Functional Memory.

The term functional memory[3,4] was used to describe the use of a special type of memory to realise various combinational and sequential functions. A cellular structure was considered in which each cell could have three possible values : 0, 1 or X (corresponding to "don't care"). A rectangular array of these cells could perform searching operations on selected fields of rows of cells and data fields from selected rows could be ORed together during output. Functional memory had the advantages of regular circuitry such as RAMs and ROMs when implemented in LSI. The approach given here does not require customized LSI.

The Design of an Associative Processor.

An associative processor was chosen as a particular example of a subset of parallel processors, the subset of all single instruction stream - multiple data stream processors. The microprogrammed control of such a processor was studied. It is an example where the data structure is considerably different from the control structure and where control problems could be acute. Associative processors are examples where much of the cost of the machine is concentrated in the control unit and other supporting modules, and where improvements in control unit design can have a significant effect. In Goodyear's STARAN processor, an array of 256 associative elements uses 2,500 integrated circuits whilst the circuits necessary to control and support this array number 6,500.[5]

There are no fundamental associative properties in the processor which was developed. These associative properties are provided by the management of the processing elements by the microprogrammed control unit. The associative operations are emulated by standard arithmetic, logical and testing operations on 16-bit word-slices of a 256-bit word, as opposed to bit-slices. The control unit could alternatively emulate a different type of parallel processor in this subclass.

A hardware realization of an 8 processing element associative processor was built using the inventory of DEC RTM modules available in this Department, with the addition of a small number of other integrated circuits.

Architectural Decisions.

It was necessary to choose applications for which the associative processor could be used whilst its operation was being studied. It was decided to consider simple programs for:

(i) air traffic control conflict detection, and
(ii) information retrieval query processing.

Both these applications are suitable for associative processing. They also seemed complementary in their use of available operations of an associative processor.

The hardware realization was constrained by the inventory of DEC RTM modules and other circuits available, and this meant that the associative processor would have 8 processing elements with each element storing 256 bits as 16 words of 16 bits. Extensive processing capability was available at the processing element level.

The associative processor is interfaced to a sequential processor - an Interdata Model 74. This stores the program for the associative processor together with the data for its processing elements, which can only be accessed in sub-blocks. Thus, the sequential processor emulates the fetch section of the ideal associative processor being considered.

There would be I/O traffic of data for PEs during associative processing which would be reduced as the number of PEs increased. The point could be reached when all the data necessary for a particular search or data processing operation is loaded in one block.

The instruction stream is provided by the sequential processor and instructions modifying this stream are trapped by the sequential processor. If the modification is conditional, it is based on status information provided by the associative processor. The associative processor microprogrammed control unit can determine whether the status information in the sequential processor requires modification and transmits this new status, or whether the new status is for local control and can remain in the associative processor status register.

The following policies were pursued:-

(a) Apart from supplying the instructions and data, the sequential processor would be free to process programs in parallel with associative processor operation.
(b) An operation or design guideline would be adopted for the hardware realization only if it could be employed effectively or even more effectively in a larger (full-scale) associative processor.
(c) By careful choice of instructions, loops would be kept at the level of the microcode where they could be handled more efficiently, thus bringing the instruction stream as close to a sequential stream as possible. This is much easier to do in an associative processor, where iteration can be handled by exploiting parallelism.
(d) Delays due to modification of the instruction stream would be minimised by careful choice of the status information to be transmitted to the sequential processor and transmission would be well before this data is required for modifying the instruction stream.

The Data Processing Structure

The data processing structure consists of eight processing elements (PEs). Each PE has 16 16-bit words, a simple ALU, two accumulators, and a means of communicating with the MCU. It is a simple data processor which could be replaced by an LSI microprocessor.

In examples of associative processors such as STARAN, serial processing of a 256-bit word allows several fields of differing length starting at differing places to be processed. Here, processing is in parallel on 16-bit words, so the data fields must be integral values of 16-bits and aligned on a 16-bit word boundary. The STARAN example offers considerably more flexibility in the layout of data. However, the necessary control is correspondingly more complex and may require a lower level of microprogramming to achieve results such as the addition of two 16-bit fields.

The possible operations of a PE can be seen from the first part of Fig. 1.

As shown in Fig. 2, the PEs are connected in a ring structure, each being able to transmit a 16-bit word to one neighbour and receive a 16-bit word from its other neighbour. In addition, the microprogram control unit (MCU) can transmit a 16-bit word to all enabled PEs. The MCU can receive the ORing of 16-bit words transmitted from all the PEs enabled. The MCU can also receive an 8-bit status word from all the PEs enabled with each PE represented by a corresponding bit of the 8-bit word.

Note that this processing structure is a general purpose parallel processor with a simple interconnection pattern and that the associative properties are provided by the MCU which alternatively could emulate a different sort of parallel processor. The interconnection structure is easily modified in the hardware realization if this should prove necessary.

The Microprogram Control Unit.

The objective is to supply minimally encoded microinstructions to both the PEs and the MCU. Here, "minimally encoded" means that the microinstruction is decoded as much as economically possible with respect to the number of control lines and the corresponding number of pins on functionally organized LSI chips such as microprocessors, RAMs and ROMs. Thus, the size of the microinstruction wordlength is not considered a restraint in the ideal case.

The requirement, in this case, is that the MCU must perform a mapping from a user instruction of 16-bits to a variable number of 64-bit microinstructions where the first 24 bits are minimally encoded for use by the PEs (the data structure) and the second 40 bits are minimally encoded for use by the MCU (the control structure). Note that the MCU microorders are at a slightly higher level in order to reduce the number of bits required in the microinstruction.

The MCU is essentially as shown on Fig. 3. Each block named SPn or RPn represents a 16*16 scratchpad memory for the select phase or read phase, respectively. These will be called functional memories because of the role that they play in the design.

The basic interpretation cycle is as follows:- The select phase register breaks up the 16-bit word stored in it into 4 4-bit fields which address 4 16-bit words in the select phase functional memories. The 4 16-bit words so accessed are ORed together and form the 16-bit word stored in the read phase register. This word in turn is broken up into 4 4-bit fields and used to address the read phase functional memories. This provides the 64 bits of the microinstruction. The first 24 bits are used to control all the PEs enabled by the enable register. The second 40-bit field controls the MCU and either generates another 16-bit word for loading the select phase register or uses the next 16-bit user instruction to load it. The same interpretation cycle continues with the only variations allowed being the way in which the next 16-bit word for the select phase register is determined.

Fig. 1 shows the format that was used for the microinstruction in the design of this associative processor. Note that a few bits are as yet unused and may have their roles assigned later.

A number of other registers, mainly self-explanatory, are shown in Fig. 3. With proper timing, one register could hold the 16-bit word for both the select phase and the read phase. The user instruction or portions thereof can be loaded directly into the select phase register. In the DEC RTM realization, the microinstruction is used directly. If this was not possible due to synchronization problems then a 64-bit master-slave register could be used with the MCU and the PEs controlled by the master flipflops whilst the slave flipflops are being prepared with the next microinstruction.

A simple example of the use of this MCU is now given. Consider that the functional memories are loaded as shown in Fig. 4. We consider the example of the user instruction 0001 0010 0111 0001 to read in a variable amount of data into a variable number of PEs. Now the user instruction is in general broken into 4 4-bit fields as follows:-

(1) First Field - the instruction operation code. For example - 0001 - read in a variable amount of data into a variable number of PEs.
(2) Second Field - any necessary information for the operation and the PEs, including data or information where data is to be found (such as data or address in the next 16-bit word of the instruction stream). For example - 0010 - load the first (2+1) scratchpad words of each PE.
(3) Third Field - any necessary information for the MCU. For example - 0111 - load (7+1) PEs consecutively. This field is stored in the X register of the MCU.
(4) Fourth Field - the control level - corresponds to a particular microprogram. This is inspected by the MCU to cause switching from one set of functional memories to another and/or loading or preloading of functional memories. For example - 0001 - the control level 1 microprogram.

Operation commences as follows:-

The fourth field is used for checking that the appropriate microprogram is loaded. The first field is loaded into the first 4-bit field of the select phase register and the second field into the second field of the select phase register. The third field is loaded into the X register of the MCU. Thus, initially the select phase register is
0001 0010 0000 0000
Only SP1 and SP2 are used during the initial user instruction interpretation. This causes the read phase register to become
0010 0000 1101 0000
Note that the one's complement of 0010 (the count of the number of words to load into a PE) has been formed in the third field. The first microinstruction is output (see Fig. 1 for the operations) and the select phase register is now loaded directly from this microinstruction with
0001 1111 1101 0000
Note that a particular bit of the microinstruction causes the first field of the select register to be loaded from the first field of the instruction. The next read phase register contents are
0010 0001 1110 0001
Note that the second field has been incremented to 1 to access the next word in the PE and that the fourth field also has this value, which will be used in the next cycle for incrementing the second field again. Note that the third field has been incremented. The test whether the right number of words have been input to a PE has been made implicit because when the third field reaches 1111, it will cause RP3 to become all zero. This will cause the user instruction to be used in loading the select phase register as described above. If the third field of the user instruction as stored in the X register of the MCU is non-zero, its value is decremented and the same instruction used again. Otherwise, the next instruction is used.

The following points need to be made about this microroutine example:-
(1) The functional memories have enough space for more instructions than was shown above. The corresponding write instruction would be
0010 0010 0111 0001
There is space for 14 other instructions which require the performing of a particular operation sequentially on a variable number of 16-bit fields of a variable number of PEs. There are many other possible methods of using the MCU of which only one example has been given here. Space restrictions do not permit a detailed description of the other microprograms for parallel and associative operations. In the above example, the minimum amount of interaction between select phase functional memories in the generation of the read phase functional memories data has taken place. It is possible for two select phase functional memories to contribute alternate bits to a 4-bit field of the read-phase register and thus allow multiway branching within the one interpretation cycle. This could prove useful in instructions where considerable decoding and decision making was being employed.
(2) Although primarily designed for associative processing, the MCU is suitable for other types of single instruction stream - multiple data stream processing. Possibly this would require some extension of the interconnection pattern considered here.
(3) In addition, it is possible to pick different modes of interpretation by suitable use of the mode of interpretation field in the microinstruction. For example, in the mode of interpretation considered above, the select phase register is loaded from the previous microinstruction except in the case when RP3 is zero. In this case, the previous instruction is used again if the X register of the MCU is non-zero, otherwise the next user instruction is loaded.

Another mode tests the activate bit in the micro-instruction and ORs the value in the X register into the SPR before the execution of the next stage of interpretation. Other modes could make direct use of the status bits in the loading of the select phase register, thus allowing much of the branching and iteration to remain at the level of the microprogram. Although there is much repetition in particular fields of the above microprogram example, which is provided for explanatory purposes, the fact remains that another microprogram when loaded can use these same fields for entirely different control purposes and with other forms of repetition. See (vi) below. Note that the incorporation of the mode of interpretation as a field in the microinstruction allows a microroutine to change the mode dynamically during user instruction interpretation.
(4) Four different microprograms are now available and can be loaded corresponding to the four possible control levels specified in the last field of the user instruction. (Sixteen possible microprograms could be specified). There would be a delay during loading. Since a user instruction never requires the use of more than one of the microprograms, if two function memory "units" were available, one could be loaded whilst the other was being used. This is especially practical since the instruction stream is close to a sequential stream, and the fetch control unit could look ahead to the last 4-bit field of the next instruction. The second functional memory unit could be loaded by looking ahead for the first instruction which does not use the current microprogram and starting to load the required microprogram in parallel with the rest of the MCU operation. When this instruction is reached, the MCU switches over to the second functional memory unit and the above process can be applied to the first functional memory unit. Having three functional memory units would take care of the worst case of user instruction branching with minimum delay. With a judicious choice of the instruction classes and the corresponding microprograms for these classes, the amount of reloading can be decreased.[6] By increasing the permitted size and number of functional memories, this problem could be eliminated altogether.

The use of the above microprogramming technique has the following additional advantages:-
(i) This form of microprogramming does not seem to be any more difficult than the microprogramming schemes of many existing machines, especially when a simulator is available. It has the advantage of postponing until late in the design process, the microprogramming stage, the specification of the control structure and the operations on it. User microprogramming although possible is not the main objective of this approach.
(ii) In considering the firmware-software interaction, this approach allows much of the software to be concentrated in the firmware. The user writes instructions at a variety of levels corresponding to the control level specified in the instruction. At present, the microprograms are used as follows:-
(1) control level 0 - resetting, load, store, shifts, tests of PE condition codes, inter-PE communication
(2) control level 1 - input and output
(3) control level 2 - addition and subtraction with provision for multiple precision arithmetic, multiplication
(4) control level 3 - general associative operations - the first field of the instruction gives the type of test. The second field gives the scratchpad location on which the test is to take place. The third field directs what will be done with the result of the test.

The user can write programs using the higher level instructions (such as the general I/O instruction and the general associative operation) descending to the lower level instructions only when the final elements of control are unavailable higher up the hierarchy (such as instructions that set particular bits in the enable register). This leads to more compact code and faster operation.
(iii) In considering the hardware-firmware interaction, many of the simple flags and small field operations that a control unit may use, such as condition and emit fields, are eliminated completely by using this particular firmware approach. In particular, tests, multiway branches, complementation, incrementing, shifting, masking and reformatting can take place implicitly. Key variables or bits of instructions can be stored and used when necessary without requiring any additional emit fields, flags or registers. Apart from the functional memories, the barest minimum of extra circuitry and registers are required.
(iv) Fault recovery can be facilitated if this is a critical factor. A special microprogram could test all the PEs in parallel, and also test the MCU. The isolation of the malfunctioning PEs can be accomplished by microprogrammed control of the enable register. Since the MCU consists mainly of identical functional memories, a spare can be switched in if one should fail. The only critical circuitry requiring consideration are the small number of MCU registers and the combinational networks. The necessary redundancy is therefore limited to these circuits and a small number of PEs and functional memories. This would be a negligible cost for a large associative processor.
(v) There is a minimum delay in processing the data structure, since by the time the data structure has been acted upon by a microinstruction, the MCU has cycled back in parallel and produced the next microinstruction. Status information when required need never come from the current microinstruction execution. This holds true even when a new user instruction interpretation has commenced. In particular, there is no delay when a test has to be made at the microprogram level, where hopefully the majority of all branches will take place.
(vi) Many instructions can use the same microinstruction fields (4 16-bit fields) and only the fields which are different require placement in a word of the read phase functional memories. The other existing fields can be used unchanged. This is one answer to the field combination problem.[4]

Microprogrammed Processors.

The previous section has attempted to show how it is possible to gain an increase in the capability of containing and processing the control structure of a processor by the addition of a minimum amount of extra complexity in the microprogram control unit. This was achieved by:-
(i) breaking up decoders for the functional memory units (or control memory in the conventional case) into a number of separate decoders,
(ii) breaking up the read phase of the control memory into a select phase and then a read phase, and
(iii) performing an ORing of the output from the select phase functional memories. This requires no additional circuitry when negative logic and open-collector drivers are used as in the DEC RTM TTL implementation.

Thus, this is one possible solution to the problem of designing processors with separate data and control structures. It would be interesting to consider this approach in different types of processors.

Microprocessors

In the design of the associative processor, each processing element was a 16-bit parallel processor with a limited number of possible low level operations. This approach was used because of availability, speed and simplicity. It simplified the microprogramming and did not require the lower level of control that a serial processor would require.

Consider however the case where a processing element is replaced with a microprocessor. This could still provide enough processing power and storage per chip if present trends continue. If microprocessors cost $20-$30 each in large quantities, then a 1K processing element associative processor would have a hardware cost of $20,000-$30,000 plus the cost of the control and supporting equipment. The fact that suitable microprocessors are available and that a possible MCU uses standard RAMs, ROMs and MSI logic circuitry means that the development costs are reduced. For example, the RCA COSMAC LSI microprocessor is very similar to the PE considered here.

Cellular Logic.

Looking further into the future and considering the continuum described by Weinberger[7], by increasing the number of decoders further, the point is reached when a decoder is addressing one of two possible output lines. The functional memory has then reached the cellular complexity of those originally considered by Flinders, et al[3]. Thus, it is possible to envisage the same sorts of operations considered here, in a cellular logic implementation with all of the many advantages covered by Kautz[8].

Acknowledgement : I am indebted to Peter Gardner for mentioning how 16 word memories could be considered as an extension of functional memories.

REFERENCES

(1) C.G. Bell and J. Grason. The Register Transfer Module Design Concept. Computer Design, May 1971, pp.87-94.
(2) P.C. Anagnostopoulos, M.J. Michel, G.H. Sockut, G.M. Stabler and A. van Dam. Computer Architecture and Instruction Set Design. AFIPS 1973 National Computer Conference Proceedings, Vol 42, pp 519-527.
(3) M. Flinders, P.L. Gardner, R.J. Llewelyn and J.S. Minshull. Functional Memory as a General Purpose Systems Technology. Technical Report T.R. 12.088, IBM United Kingdom Laboratories Ltd, Hursley Park, Winchester Hampshire, July 1970.
(4) P.L. Gardner. Functional Memory and its Microprogramming Implications. IEEE Trans. on Comp., Vol. C-20, No. 7 (July 1971) pp.764-775
(5) J.D. Feldman and O.A. Reimann. RADCAP : An Operational Parallel Processing Facility. AFIPS 1974 National Computer Conference Proceedings, Vol. 43, pp. 7-15.
(6) C.C. Foster and R. Gonter. Conditional Interpretation of Operation Codes. IEEE Trans. on Comp, Vol. C-20, No. 2, (January 1971), pp. 108-111.
(7) A. Weinberger. Hybrid Associative Memory Concept. Computer Design, January 1971.
(8) W.H. Kautz. Cellular Logic-in-Memory Arrays. Trans. on Comp., Vol. C-18, No. 8, (August 1969), pp. 719-727.