THE MICROPROGRAMMED CONTROLLER CONCEPT

Kenneth J. Thurber[1,2]
and
Glen R. Kregness[1]

ABSTRACT

This paper describes the MPC (Microprogrammed Controller) Concept used at Sperry Univac Defense Systems Division to implement real-time computer emulations. It discusses the concept and reasons for microprogrammed emulation and the basic MPC approach. Enhancements developed for the MPC and their impact are also discussed.

INTRODUCTION

Microprogramming has been a control logic implementation technique that conceptually has been available to designers since it was first introduced by Wilkes in the early 50's.[1] However, it was not until the mid-60's when the technique became popular and in widespread use.[2]

Emulation of one computer system by another computer system conceptually has also been a designer's tool for many years. Some "emulators" are hardwired copies of early machines.[3] Other emulators consist of "software packages" which simulate a specific computer system.[4]

Recently, the use of microprogramming techniques to implement an emulation of an alternative computer system has been a prevelant emulation technique.[5,6,7,8,9,10] This paper discusses the concept, evolution, and use of the MPC (Microprogrammed Controller) concept used at Sperry Univac Defense Systems to implement computer emulations.

To date, this concept has been applied to, and emulators built for, over 20 alternative repertoires or alternative performance levels of a specific repertoire (ranging from the AN/UYK-20(V) to the Honeywell DDP-24). For any given repertoire we have produced a variable number of machines. In some cases, we have built only one copy of a specific emulator; in other cases, we have produced over 700 copies of a specific emulator.

In the literature, various designers have argued for vertical microprogramming (encoded microinstructions), horizontal microprogramming (non-coded microinstructions), hybrid schemes (using two levels of microcode: one vertical and one horizontal), and various combinations of the above techniques. The MPC can be viewed as a vertically microprogrammed machine with the ability to call horizontally encoded subroutines which are supported by easily tailorable hardware. This concept is at the same time elegant, yet very hardware cost effective.

[1] Sperry Univac
St. Paul, Minnesota

[2] Computer Science Department
University of Minnesota
Minneapolis, Minnesota

WHY MICROPROGRAMMING?

Microprogramming is a technique for implementation of the control logic of a digital system. Digital systems are made up of registers and networks interconnected to perform a given function. In hardwired machines, some registers are addressable because they may be directly specified in an instruction. Adders, shift networks, and data busses are not generally controlled by an instruction. The assembly language programmer may not even be aware that this hardware exists.

Microprogramming normally treats all registers and networks as addressable within the microinstruction format. The control logic therefore becomes the microinstruction sequence. Timing is controlled by microinstruction sequences.

Microprogramming must be intimately concerned with the digital hardware to be efficient. The microinstruction format will be heavily influenced by currently available semiconductor integrated circuits or the design of custom semiconductor devices.

Emulation becomes the process of writing a microprogram which, when run on a microprogrammed computer, produces the same results as the original machine. Generally, most authors ignore timing issues when discussing emulation. However, in a real-time environment, timing considerations must also be considered.

There are a number of advantages to implementing an emulator using microprogramming:

1) flexibility,
2) ease of alteration,
3) provides a basic building block which can be altered to emulate other computers,
4) simplifies logistics,
5) lower recurring and nonrecurring costs,
6) can take advantage of semiconductor advances, and
7) shorter development time.

Classical microprogramming (horizontal: non-encoded) usually dedicates bits in the instruction for direct control over registers and logic networks; e.g., a register having parallel load capability would have a bit within the instruction word reserved for controlling this specific function. This approach generally leads to wide microinstructions.

Firmware microprogramming (vertical: encoded), on the other hand, takes the opposite approach; i.e., functions are encoded. The firmware approach closely resembles conventional hardwired design techniques. A "narrow" instruction word is implemented with encoded fields in the microinstruction assigned to specific functions; e.g., Load, Add, Subtract, or Shift. In fact, many classical microprogramming people do not consider firware as a microprogramming approach at all, but rather an extremely simple computer. The hybrid approach combines the two approaches for memory and execution efficiency with one vertical microinstruction pointing to one or more horizontal microinstructions.

The main advantages of vertical microprogramming are:

1) efficiency of micromemory use (low cost)
2) only useful functions are available,
3) ease of programming, and
4) ease of software-support tool generation.

The main advantages of horizontal microprogramming are:

1) parallelism,
2) high performance, and
3) low number of instructions necessary to emulate another computer's instruction.

The hybrid approach leads to the advantages of both vertical and horizontal schemes.

Further, it is possible that a single microprogrammed controller can emulate many different architectures with suitable microcode changes.

THE MPC CONCEPT

The concept of the MPC (Figure 1) is quite simple. It revolves around a standard building block: the Microprogrammed Controller. The MPC (Figure 1, Figure 2, Table 1) is used in the AN/UYK-20. This is a simple, vertically encoded, register-oriented machine. This provides the advantages of low cost and flexibility. Specific microinstructions can point to a second-level microinstruction, which is analogous to an instruction which acts as a nanoinstruction would act; however, it acts

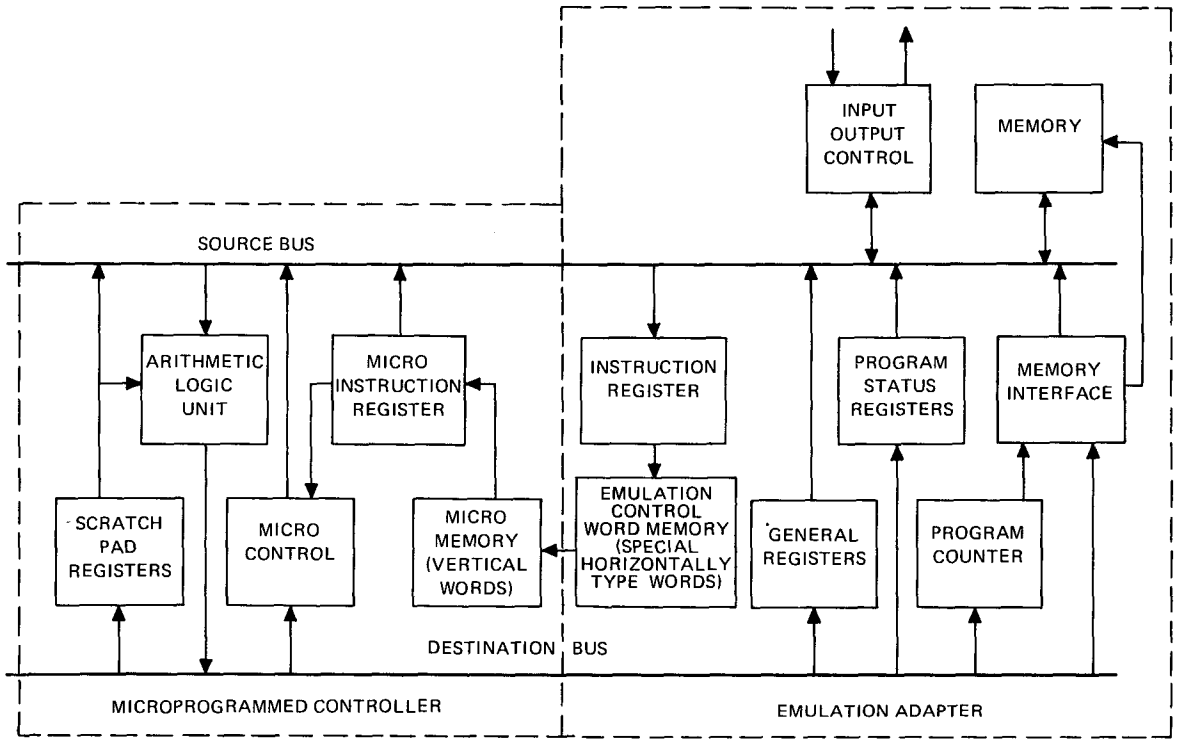

Figure 1. Functional Architecture

15	14	13	12	11	10	9	8 7 6 5 4 3 2 1 0
							EMULATION POINTER ADDRESS ONE'S COMPLEMENT
			M	I	O	U	CONTROL FUNCTION FROM EB1 OR EB2 (EMULATE BRANCH 1 OR 2)**
			0	0	0	0	RR UNARY – MODIFY ECW POINTER BITS 4–1 BY M* – NO OVERLAP***
			0	0	0	1	RR NO OVERLAP
			0	0	1	0	RR UNARY – MODIFY ECW POINTER BITS 4–1 BY M* – OVERLAP
			0	0	1	1	RR OVERLAP
			0	1	0	0	RK UNARY – MODIFY ECW POINTER BITS 4–1 BY M* – NO OVERLAP – BRANCH 1 TO INTERIM SEQUENCE
			0	1	0	1	RK NO OVERLAP – BRANCH 1 TO INTERIM SEQUENCE
			0	1	1	0	RK UNARY – MODIFY ECW POINTER BITS 4–1 BY M* – OVERLAP – BRANCH 1 TO INTERIM SEQUENCE
			0	1	1	1	RK OVERLAP – BRANCH 1 TO INTERIM SEQUENCE
			1	0	0	0	RI INHIBIT NEXT INSTRUCTION WRITE
			1	0	0	1	RI ENABLE NEXT INSTRUCTION WRITE
			1	0	1	0	UNDEFINED
			1	0	1	1	UNDEFINED
			1	1	0	0	RX INHIBIT NEXT INSTRUCTION WRITE – BRANCH 1 TO INTERIM SEQUENCE
			1	1	0	1	RX ENABLE NEXT INSTRUCTION WRITE – BRANCH 1 TO INTERIM SEQUENCE
			1	1	1	0	RX INHIBIT INDEXING – INHIBIT NEXT INSTRUCTION WRITE – BRANCH 1 TO INTERIM SEQUENCE
			1	1	1	1	RX INHIBIT INDEXING – ENABLE NEXT INSTRUCTION WRITE – BRANCH 1 TO INTERIM SEQUENCE

			OPERAND MEMORY MODE
0	0	0	READ (FULL WORD)
0	0	1	WRITE (FULL WORD)
0	1	0	READ ODD WORD
0	1	1	WRITE ODD WORD
1	0	0	SPLIT CYCLE (i.e., READ, MODIFY, WRITE)
1	0	1	WRITE 0
1	1	0	READ THE BYTE SPECIFIED BY BIT 15 OF THE CONDITION REGISTER
1	1	1	WRITE THE BYTE SPECIFIED BY BIT 15 OF THE CONDITION REGISTER

*M → ONE'S COMPLEMENT OF M
**EB1 AND EB2 CONTROL ACCESS TO INTERIM ROUTINES
***RR, RK, RI, RX ARE INSTRUCTION ADDRESS FORMATS

Figure 2. Emulator Control Word

more like a subroutine. Further, this second-level instruction has as its goal the performance of the functions which are specific and unique to an architecture which is being emulated. For this reason, this portion of the machine is known as the emulation adaptor. Some functions of the emulation adaptor may be hardwired. Typical functions performed by the emulation adaptor include format decomposition, status setting, control, and interpretation, operand fetch according to memory mode designation, etc. The emulation adaptor could be 1) built from programmable logic, 2) built as a MPC with tailored firmware, etc. We have chosen (to date) to build our emulation adaptors from hardware due to speed considerations and the fact that generally we are attempting to emulate only a single architecture.

The microinstruction repertoire is given in Table 1. The ECW (emulation control word) format is shown in Figure 2 (for the AN/UYK-20). The emulate microcommand causes an ECW to be read and its specified conditions to be met. Thus, the sequence of events is:

<Instruction> <User>
is fetched and interpreted by

<Emulation Adaptor> <Emulation Tailoring>
provided for emulation by hardware designer uses "interim sequences" to decompose the instruction, fetch operands, and prepare the instruction for transmittal to the MPC for execution

<MPC> <Register Oriented Machine>
programmed by microcode (hardware) designer to act upon data placed into MPC scratch pad registers by the emulation adaptor

<Result> <User>
answer from scratch pad registers is placed into appropriate positions in emulation adaptor

The interim sequences are sequences (possible hardwired) which provide for operand fetch, format decomposition, etc. Obviously, the most efficient design will minimize the changes required to an emulation adaptor to change it from an emulator of machine A to emulation of machine B.

Further, the ECW may be viewed as either:

1) A horizontal microinstruction,
2) An extension of the vertical microinstruction, or
3) A hardwired interrupt-oriented aid to the vertical microinstruction which calls in the emulation adaptor hardware to perform emulation specific functions.

Table 1. Microinstruction Repertoire

Instruction Format				Description
15-12	11-8	7-4	3-0	
F = 00	D	S	M	Transfer
F = 01	X			Unconditional Branch
F = 02	D	S	M	Add S2
F = 03	D	S	M	Shift
F = 04	D	S	M	Add S1
F = 05	D	S	M	Subtract
F = 06	D	S	M	Logic I
F = 07	D	S	M	Logic II
F = 10	D	K		Add Constant
F = 11	D	K		Subtract Constant
F = 12	D	K		Transfer Constant to D1
F = 13	D	K		Transfer Constant to D2
F = 14	FII	K		Branch
F = 15	FII	S	M	Micro Control
F = 16	FII	K		Micro Repeat
F = 17	D	S	M	Emulate

Definition of fields

F	Function code
FII	Subfunction code
D	Destination designator field
S	Source designator
X	Branch address designator
M	Sub-function or modifier designator
K	Constant - 8-bit absolute value or subfunction code

Although it is possible to make the emulation adaptor and the MPC micromemory accessible to the user, to date we have not provided this capability in our machines since the ECW and micromemory are built from read-only memories.

The previous discussion assumes no overlap of microinstructions or parallelism in the MPC. In the MPC II, for example, the microinstructions are executed in a three-stage pipeline. Further, the "emulate start" command is the "last instruction" of a macro and it causes the next macro to be read and its interim sequences to be started. Thus in general the microcode for a typical macroinstruction takes the form:

1) Interim Sequences (if any: specified by EB1, EB2) — parse instructions: obtain operands
2) Vertical Microcode — execution
3) Emulate Start (fetches next macro) — start next macro

MPC ENHANCEMENTS

Since the development of MPC I we have developed an enhanced version of the MPC (known as MPC II). In addition, LSI versions of the MPC II (MPC III and MPC IV) are available. The MPC II instruction format has some minor modifications made to it to obtain the MPC III and MPC IV architectures. MPC IV and MPC II are identical (functionally) although the hardware structure is different. MPC III has more bits in the F and M (function and function modifier) fields due to the necessity to provide more function bits to control the AMD 2901 chip. The characteristics of these MPCs are summarized in Figure 3. The enhancements made to MPC I were based upon considerable analysis work which is summarized in the following paragraphs.

The emulation adaptor concept is the primary feature which has yielded a decided performance edge to the MPC in contrast to other emulation approaches. The emulation adaptor is an address generator which converts the MPC into an interrupt-driven system. Its primary function is to dynamically monitor the emulated machine state and produce starting addresses of microinstruction subroutines which implement the desired functions.

	MPC I	MPC II	MPC III	MPC IV
NUMBER OF BUSES	2	4	3(2)	3
NUMBER OF MICROINSTRUCTIONS	16	16	16	16
MICROINSTRUCTION LENGTH (BITS)	16	36	40	36
CONTROL STORE TYPE	ROM	ROM/RAM	ROM/RAM	ROM/RAM
MAXIMUM CONTROL STORE SIZE	4K X 16	4K X 72	4K X 80	4K X 72
MICRO-NESTING LEVELS	1	4	4	4
ARITHMETIC TYPES	2'S, 1'S	2'S, 1'S	2'S, 1'S	2'S, 1'S
HARDWARE REQUIRED	REFERENCE	SAME AS MPC I	2/3 MPC	2/3 MPC
CIRCUITS USED	GENERAL TTL	GENERAL ECL 10K	AMD 2901, AMD 2909, LSTTL PARTS	PLA'S, LS 181 ALU SLICE, AMD 2909, AMD 2914, LSTTL PARTS

Figure 3. MPC Attributes

In the MPC, the emulation adaptor is probed by means of a special microinstruction which serves as a trigger to generate the address of the next routine to be entered. This particular function is not totally unique to the MPC; however, few machines carry this function to the extent that the MPC does. One feature of the MPC which is unique is the supplementary control which is supplied by the emulation adaptor. Such things as main memory cycle modes, interim-sequencing and address-generation characteristics, general register selection, macro-instruction overlapping, and condition code setting are accomplished and interpreted through the emulation adaptor. In effect, the emulate instruction temporarily transfers machine control to the emulation adaptor which then issues control signals to the rest of the machine in addition to supplying the branch address of the next microinstruction routine.

The major improvements incorporated into the MPC II reflect the extensive use of the emulate instruction, which is deleted as a special instruction and encoded as a field in every microinstruction, as well as the addition of facilities to relieve the following limiting characteristics of MPC I:

1) limited register addressing
2) limited accumulator selection
3) limited conditional branch capability
4) difficulty of introducing constants
5) lack of parallel control capability.

The main differences between MPC II, MPC III, and MPC IV and MPC I can be seen by examining the MPC II instruction format (Figure 4) and the MPC I instruction format (Table 1). Both machines have function (f) and function modifier capabilities (M). The instruction repertoires (Figure 4 and Table 1) differ mainly in that the emulate command (MPC I) has become a field in the MPC II (E-field). Further, the MPC II has added instruction parity (P) and an overlapped even/odd microinstruction branch capability (B). The B Field allows the microcontrol store to be viewed as 4K x 32-bits or 8K x 36 bits, and this feature can be exploited in microinstruction branching during overlapped operations. A trigger field (T) was also added so that specific conditions and/or functions can be programmed for use in obtaining microinstruction parallelism; e.g., it is conceivable to initiate memory during an add microinstruction. An additional source bus (SX) and a qualifier (SXE) were added to MPC II for flexibility reasons. Additionally, a number of microsubroutine nesting levels were added to MPC I to facilitate more efficient microprogramming. The resulting MPC II block diagram is shown in Figure 5. The system has also been equipped with a remote bus for maintenance purposes and the ability to have a constant ROM for storage of arithmetic constants. The ECW for the MPC II is shown in Figure 6 and the instruction repertoire in Figure 7. The ECW of MPC I and MPC II are similar. They both contain a microcode address; however, the MPC II contains an E/O bit to specify even/odd word addressing of the microstore. Both MPCs contain a memory operand mode control field. In MPC II the DP field is used for general register selection. The remaining fields of MPC II provide for format sensing (F2, F1), memory enable (M), interim sequence control (I), overlap of next instruction readup (0), and unary operation designator (U). These functions were basically encoded in the control functions of the MPC I ECW.

The above identified changes were basically determined through our experiences building emulators based upon the MPC I architecture. The three variations of the MPC II architecture are due to detailed cost/performance issues involved with parts selection to meet specific equipment requirements.

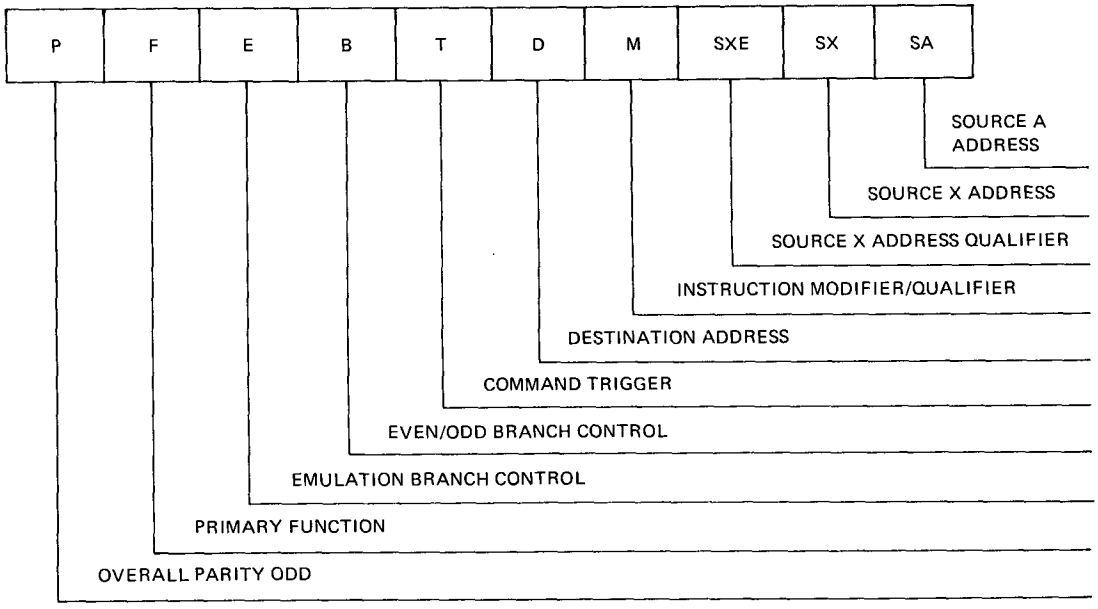

Figure 4. Basic MPC II Microinstruction Format

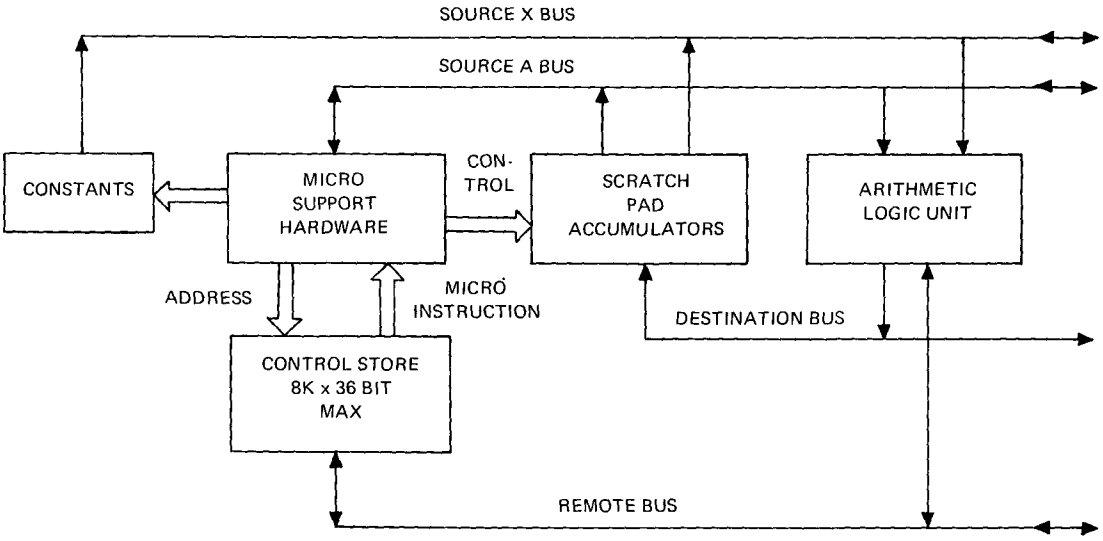

Figure 5. MPC II Block Diagram

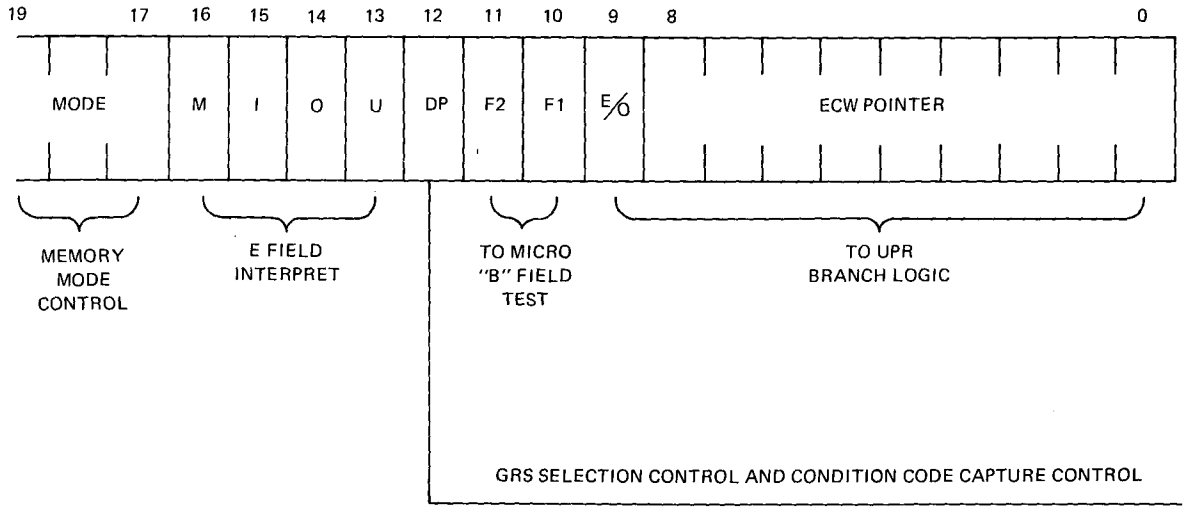

Figure 6. Emulation Control Word Format

FUNCTION CODE	DESCRIPTION	MNENOMIC
00	LOAD CONSTANT	LDK
01	BRANCH	JP
02	MICRO CONTROL	MCL
03	REPEAT	RPT
04	SHIFT	SH
05	TRANSFER REMOTE	TXR
06	TRANSFER CONTROL STORE	TXC
07	UNASSIGNED	
10	ADD	ADD
11	SUBTRACT	SUB
12	LOGICAL I	
13	LOGICAL II	
14	CONDITIONAL TRANSFER SA	CTA
15	CONDITIONAL TRANSFER SX	CTX
16	CONDITIONAL SUM SA	CSA
17	CONDITIONAL SUM SX	CSX

Figure 7. MPC II Microrepertoire

CONCLUSION

The MPC concept has been used extensively. Currently, we have emulated 12 different repertoires or alternative performance levels of a specific repertoire with the MPC I, two with the MPC II, and seven with the MPC III and three with the MPC IV. In all, we have produced over 1000 machines which contain an emulator based upon the MPC concept. We have emulated machines as diverse as the Honeywell DDP-24 (training center 24-bit word length), the CP890 (shipboard 30-bit word length), the SKC 2070 (32-bit word length; airborne radiation hardened computer) and the AN/UYK-20 (16-bit word length; shipboard) computers.

We feel that the MPC concept provides a good cost/performance trade-off for the development of real-time architecture emulators. The principles upon which the MPC achieves its success are primarily involved with the definition and efficient use of the emulation adaptor to provide accurate emulation of complex systems.

REFERENCES

[1] M. V. Wilkes, "The Best Way to Design an Automatic Calculating Machine," Manchester University Computer Inaugural Conference, Ferranti Ltd., London, England, 1951, pp. 16-21.

[2] M. A. McCormack, *et al.*, "1401 Compatibility Feature on the IBM System/360 Model 30," *CACM*, August 1965.

[3] Sperry Univac, "Univac 1832, Avionics Computer (Single and Multiprocessor), General Description" (Univac PX 5627A).

[4] R. I. Benjamin, "The Spectra 70/45 Emulator for the RCA 301," *CACM*, December 1965.

[5] Sperry Univac, "AN/UYK-20: Technical Description" (Univac PX 10431C), November 1976.

[6] W. T. Wilner, "Microprogramming Environment on the Burroughs B1700," *COMPCON '72.*

[7] Nanodata Corporation, "QM-1 Hardware Level User's Manual," 1976.

[8] Control Data Corporation, "Control Data 5600 Series of Microprogrammable Processors: Reference Manual," 1972.

[9] C. Neuhauser, "An Emulation ORIENTED Dynamic Microprogrammable Processor (Version 3)," Stanford University.

[10] C. V. Ramamoorthy, "A Survey of the Status of Microprogramming," *Advances in Information Sciences* (ed. J. T. Tou), Volume 5, 1974.