SIMULATION OF MICROPROCESSOR OPERATION FOR PROGRAM DEVELOPMENT AND CHECKOUT

D.L. TRIMBLE and B.E. PETRASKO
Electrical Engineering and
Communication Sciences Department
Florida Technological University

INTRODUCTION

This paper is part of work done for the National Science Foundation under Grant GK 42071 titled "Investigation of the application of a hardware parsed recursive string processing language to graphical systems." The aim of this grant is to firmware parse a certain class of input strings in the development of an extensible string processing language similar to text reckoning and compiling (TRAC).[1] During this grant, a full operating language called TOSCL was developed to run on a Data General NOVA 1220.[2,3] The second phase was to implement the parse on the Intel 3000 system.

The primary aim of this paper is to describe the implementation of microprograms using a sixteen bit minicomputer and to show some of the software developed to support microcoding, assembling of microcode, loading the microcontrol store, and the debugging of the microprograms.

The development of Intel's Schottky Bipolar LSI microcomputer elements has brought microprogramming to the field of microprocessors. These devices allow almost infinite variety of design applications, control word size, and control word configuration. This versatility makes the development of support software a difficult if not impossible task.

The emphasis in design and construction of the TOSCL microprogramming system was to be able to microprogram with the least amount of effort. Since the major programming effort was to be the development of TOSCL, only a minimum of software was developed. The minimum support required was (1) a symbolic editor, (2) a microcode assembler, (3) a loader for microcontrol store, and (4) microprogramming debugging aides. This paper shows how these programs combined with hardware functions allowed simulation and debugging of the Intel 3000 microprocessor system.

THE MICROCONTROL WORD

The minicomputer supporting the TOSCL system is a Data General NOVA 1220, and since this is a sixteen bit machine, a control store size of sixteen bits is desirable. The 3000 system configuration suggested by Intel uses an eighteen-plus bit control word (see Figure 1a). To implement a sixteen bit control word, it was necessary to give up some flexibility in the system. After looking into the application which is string interpretation, the facilities considered least critical are portions of the flag control and branching (jumpcode) facilities.

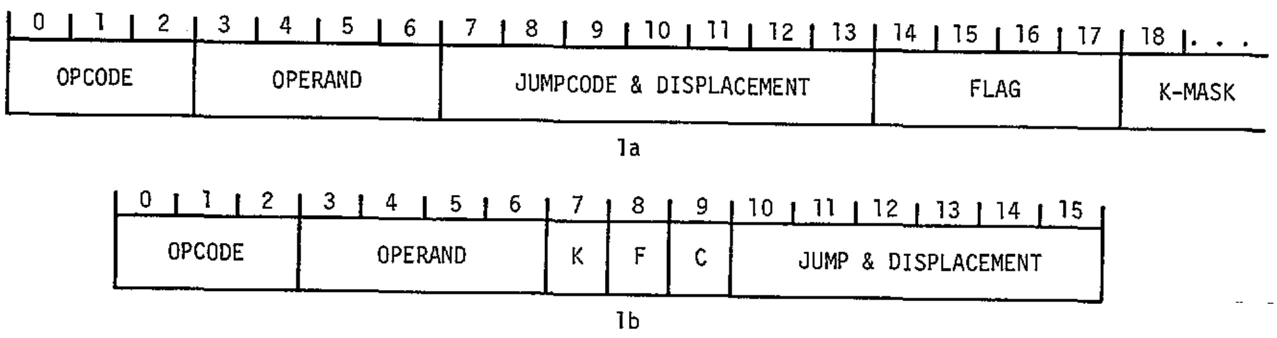

Figure 1. Microcontrol Word

The resulting control word is sixteen bits (see Figure 1b). It represents a loss of four jump instructions and individual control of one flag bit, the Z-flag bit. The K-buss mask can only be set to all one or all zero.

The most often used 3000 branch command uses the PX inputs of the 3000 system to force a jump in control store. Since this major decision process is accomplished through a hardware decode, the use of one flag control bit is all that is necessary. A reduction in the jump set could be accomplished with the loss of the use of one of the flag functions. Random logic has been added to accommodate this change in the control word format. The 3000 control word jump instruction set is hardware decoded to recover the bits removed by the change in the control word. The I/O signals are also decoded from the NOP Instructions and its operand (function field) to accomplish the required input, output, and memory control.

The resulting control word is sixteen bits (see Figure 1b). It represents a loss of four jump instructions and individual control of the flags.

THE ASSEMBLER

The Intel 3000 system allows such a variety of configurations that the resulting generalized assemblers are often difficult to use, hard to transfer between machines, impossible to implement on small systems, and require large amounts of computer time.

The concept developed was not to generate a general assembler but to use an existing assembler with macro capabilities to define an instruction set which includes both Intel instructions and instructions unique to this task. This loss of portability due to the omission of four jump instructions and the control of the Z-flag was examined and found to be not sufficient to warrant additional efforts. This use of a macroassembler to generate a cross-assembler through the definition of a set of instructions is an efficient approach but results in a highly machine dependent cross-assembler.

The cross-assembler developed to assemble programs for the TOSCL system is run on Data General's macroassembler and has the following format:

```
(LABLE:) OPCODE(H) (1) OPERAND JUMPCODE
  DISPLACEMENT
```

The portion shown in parenthesis is optional. The H appended to the opcode controls the hold of the carry flag. The 1 is used to set the flag output (FO) which is used as the carry input (CI). The label and comment fields are aides for the programmer to use in documentation.

The operand field specifies the range of the opcode and the K-buss information. The operands are broken down into three register groups: (1) register operations, (2) memory operations, and (3) input operations.

The jumpcode and displacement are entirely independent of the opcode and operand. Therefore, the programming techniques for this system are based on established microprogramming techniques.

The output from the macroassembler is in a special binary format. This format can be reduced to a core image file using the relocatable loader. This software along with the symbolic editor is part of the standard system and requires no modification. The resulting core image file can either be executed from the host systems memory or can be loaded into the control store (high-speed random access memory (RAM)) that appears to the 3000 system as a read only memory (ROM).

TOSCL SYSTEM HARDWARE

The current design of the Intel 3000 system used in the TOSCL System is shown in Figure 2. This system uses Intel's suggested system architecture.[5,6] The RAM for the 3000 system is the host system memory and is accessed using direct memory access (DMA). This memory is addressed using the A-outputs of the CPE array, while the D-outputs are used for the write data, and the M-inputs are used to read memory. The DMA operation is automatically initiated by decoding certain 3000 commands (LMM, LMI, LMMH, and LMMIH). If a previous DMA request has not been completed, the instruction will be deferred by halting the clock until the operation is complete. Also to be sure that the microprogram will not try to use DMA results prematurely, the system has a wait-for-memory command (WAIT). The system allows for computation that does not effect significant registers to be performed while the DMA operations are being completed. This may take from 1500 to 3500 microseconds.

The clock circuit can be run in two basic modes, free running and single step. The free running mode once enabled will run until it is halted. The halt can be generated by one of two commands. Done sets the done flag of the host system, and INTR generates a software interrupt to the host system. The clock is also temporarily halted by the second DMA request until the first has been completed. The second mode of the clock is single step. This allows for the execution of one microinstruction at a time and is used for tracing and debugging programs. The system, when in single step mode, uses the main memory as the source for the microinstructions. For normal operation, the control store memory (256 x 16 bits) is used as the source of microinstructions. This memory can be loaded directly from a core image file which was generated by the cross-assembler and the relocatable loader.

The contents of the control store memory can be examined, modified, or verified under program control.

The local memory (16 x 16 bits RAM) is used to jam instructions into the 3000 system. Through the jamming of the proper instructions, all of the registers may be examined. Also the control store address may be examined, along with the contents of control store.

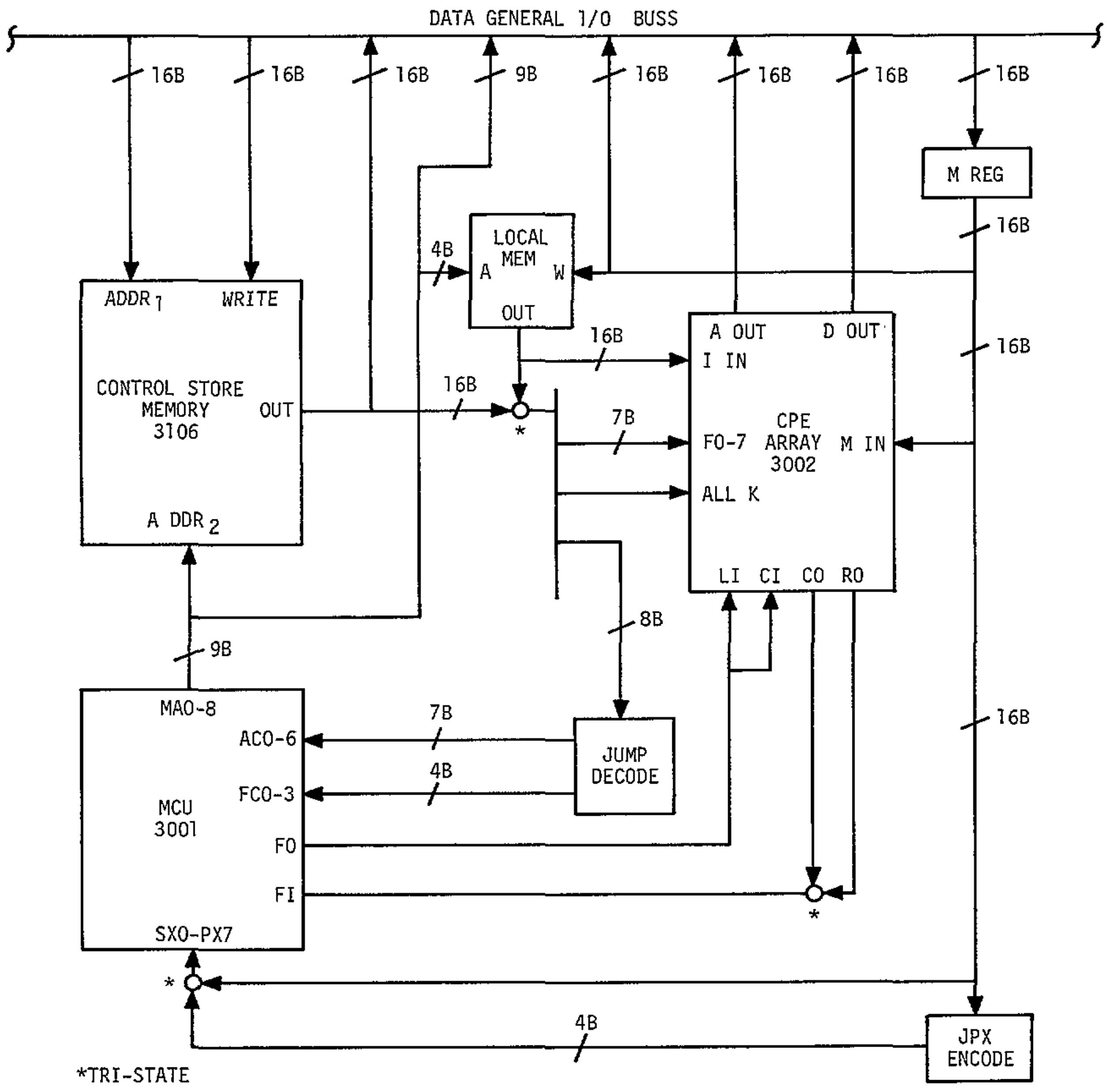

Figure 2. TOSCL System Diagram

MEM OUT	10	11	12	13	14	15	MA	6	5	4	3	2	1	0
JCC	0	0	D3	D2	D1	D0		0	0	0	D3	D2	D1	D0
JCR	0	1	D3	D2	D1	D0		0	1	1	D3	D2	D1	D0
JCF	1	0	0	D2	D1	D0		1	0	1	0	D2	D1	D0
JFL	1	0	1	D2	D1	D0		1	0	0	0	D2	D1	D0
JZR	1	1	0	D2	D1	D0		0	1	0	0	D2	D1	D0
JRL	1	1	1	1	D1	D0		1	1	1	1	1	D1	D0
JPX	1	1	1	0	D1	D0		1	1	1	1	0	D1	D0

Figure 3. Jump Set and Decode

The jump decode operation is shown in Figure 3, above. The resulting jump set, as has been discussed, is reduced from the original Intel jump set, but was found to accomplish the desired operations with no difficulty.

The JPX encode is used to detect and encode privileged characters. The input to the JPX encode is the M register which is the Memory Buffer Register of the TOSCL system. The word in the M register is then encoded and used to force a jump in control store based on the character in the M register. The M register also can be used to force the first address in control store through the use of the LOAD input of the memory control unit.

The system has inputs to and from the host computer which are used for the simulation of the final system. The final system would also use its own memory thus speeding overall system operation.

SUMMARY AND COMMENTS

The development of microprocessing system simulators has been typically based on a cross-assembler and simulator package designed to run on a large computer system. The program developed on the simulator is then loaded into a separate microprocessor based system which is used to support the hardware design phase.

The effort at Florida Technological University has resulted in an integrated software and hardware support system. It uses standard minicomputer software (macroassembler, loader, editor, etc.) and interfacing facilities to provide not only a powerful development phase tool but also a base for evaluation and training.

Evaluation can be accomplished by adding hardware to gather run time statistics which can then be processed by the minicomputer system. Training can be accomplished using a combination of computer-aided instruction and real-time operation. Both of these "side effects" are presently being investigated.

One final comment on the technique used for microprocessor system simulation; very often a great deal of cost and/or effort is spent on obtaining design and development tools which are essentially a one-time effort. If a minicomputer is available, it can be used as a base for software development aimed at a general system that can be used with the microprocessor that is best suited for the individual application. It appears that this approach is the straightforward and cost-effective means of system design.

REFERENCES

1. Mooers, C. N. and Deutsch, L.P., "TRAC a Text-Handling Language," Proc. ACM 20TH National Conference, 1965, pp. 229-246.

2. Petrasko, B. E., "TOSCL - A Function Generator Language for CAI System Implementation," Doctoral Dissertation, Electrical Engineering Department, The University of Detroit, 1973.

3. Petrasko, B. E., "Hardware String Processing Using Schottky Bipolar LSI Microprocessing," Proc. of 1976 IEEE Southeastern Conference and Exhibit, pp. 185-189.

4. Rattne, J., Cornet, J., and Hoff, M.E., "Bipolar LSI Computing Elements Usher In a New Era of Digital Design," Electronics, September 5, 1974.

5. "3001 Microprogramming Control Unit," Intel Corp., 1974.

6. "3002 Central Processing Element," Intel Corp., 1974.

ABOUT THE AUTHORS

MR. DAVID L. TRIMBLE is a graduate student in the Department of Electrical Engineering and Communication Science at Florida Technological University. His responsibilities are in the areas of animation and control systems. He has extensive experience in minicomputer and microcomputer systems and is employed at AACTS in Orlando. He holds a B.S. degree in engineering and is a member of the Institute of Electrical and Electronic Engineers.

DR. BRIAN E. PETRASKO is an Assistant Professor of Engineering Science in the Department of Electrical Engineering and Communication Science at Florida Technological University. He has been affiliated with I.B.M., G.E., Ford, Edutronics, Martin-Marietta, and the Naval Training Equipment Center. He has worked and published in the areas of computer assisted instruction, computer aided design, computer architecture, and microprocessor design and applications. He holds the B.E.E., M.E., and Doctor of Science degrees from the University of Detroit. He is a member of the Institute of Electrical and Electronic Engineers and the Association for Computing Machinery.