24 Sep 1973

Memory Utilization for a Dynamically Microprogrammed Computer

Paul D. Stigall
Missouri University of Science and Technology, tigall@mst.edu

MEMORY UTILIZATION FOR A DYNAMICALLY MICROPROGRAMMED COMPUTER*

Paul D. Stigall
Department of Electrical Engineering and Computer Science
University of Missouri-Rolla, Rolla, Missouri 65401

Abstract: A particular, dynamically microprogrammed computer (proposed by Tucker and Flynn in Commun. of ACM, April 1971) is considered with respect to main memory and micromemory utilization. A dependency is shown between memory utilization and utilization of the arithmetic and logic unit.

An equation is derived to express the average microinstruction execution time in micromemory cycles. From this equation, it has been determined that for an example representing instruction buffering and branch anticipation will reduce the average execution time by 5 percent, while a good data management or interleaving memory modules example will decrease it by 59 percent.

Introduction

Microprogramming covers a broad spectrum of computer interests, such as hardware and systems, logic design, languages, and simulation. Thus an all inclusive definition of microprogramming is extensive and detailed (1). This study considers two important properties of microprogrammed systems: fast storage (2) and simple logic processing units. Traditionally, fast storage has consisted of read-only memories (1)(3), while more recent memory developments have stimulated dynamically microprogrammed systems (4)(5) which use fast read-write memories.

In many respects, the static microprogrammed system can be considered a subset of the dynamic case. That is, a static system consists of one fixed set of microinstructions, whereas, a dynamic system can consist of different sets. Hopefully, the dynamic aspect will be helpful in leading to a more flexible computer architecture and more efficient interfaces between hardware and software.

The structure and memory utilization considered in this paper is for the Tucker-Flynn (5), dynamically microprogrammed processor. This processor is a sequential, instruction-driven computer with two levels of read-write memory, main memory and micromemory. The micromemory speed is approximately ten times faster than the main memory.

*(This work was supported in part by NSF Grant GJ-32596.)

Tucker-Flynn Processor (5)

The Tucker-Flynn, dynamically microprogrammed processor is differentiated from other processors by a uniform addressing scheme for the main memory and micromemory. Each memory consists of 64-bit words with the micromemory memory being limited to 4K in size. The accumulator, instruction address register, 16 general purpose registers, and 16 data mask registers are addressed as part of the micromemory, which includes them in the common addressing scheme.

The arithmetic and logic portion is composed of a two-input, 64-bit, parallel adder and a single input shifter/masker. The shifting--masking occurs on the output of the shifter--of data is performed in parallel with data addition.

Because the microinstruction format is horizontal or minimally-encoded, it permits an explicit view of the data use. The 64-bit format, which is broken into specific fields, each with its own interpretation. The first 32 bits indicate the control gatings necessary to execute the operation, whereas, the last 32 bits specify two operand fields.

Microinstruction Execution

One method of viewing the execution of a microinstruction for the Tucker-Flynn processor is in terms of accesses to main memory, micromemory, and special words as well as in the arithmetic and logical operations. This view is presented in Fig. 1, in which critical dependency is indicated either by adjacency, an arrow, or both.

The broken lines indicate that either main memory or micromemory can be accessed. Special words, like the assumulator, are initially assumed to have the same access time as micromemory. Also, sequencing, address calculation, and arithmetic and logical execution are assumed to each take one micromemory cycle. At this point, no consideration has been given to multiple accesses to memory modules or to simultaneous addititions.

Fig. 2 is a simplification of Fig. 1 and is based on the assumption that the microinstruction address register, the accumulator, the general purpose registers, and the data mask registers are implemented by high speed registers with simultaneous outputs available. The addressing structure of these registers would still be common with main memory and micromemory. No consideration at this point has been given to multiple accesses to main memory and micromemory or to simultaneous additions.

Fig. 3 introduces the restrictions of single additions and single accesses. That is, only one addition can be performed at a time and only one word, in both main memory and micromemory, can be addressed at a time. One observation from Fig. 3 is that after the initial microinstruction has been executed, the micromemory and adder are utilized continuously except for accesses to main memory. That is, micromemory and adder utilization would be 100 percent with accesses only to micromemory except for the first and last microinstruction. If an access to main memory occurs, micromemory would be completely idle, and the adder would be idle for a period equal to the difference between the main memory and micromemory cycle time. This still assumes that the addition time is similar to the micromemory cycle time.

The average microinstruction execution time (AET) measured in the micromemory cycles can be determined from Fig. 3 as follows:

$$AET = \frac{(NMA \cdot R + 2 \cdot N1A + N2A) \cdot MAT}{N \cdot MIT} + \frac{NMI + 2 \cdot N1I + N2I}{N}$$

where N = number of microinstructions
NMA = number of microinstructions fetched from main memory
R = improvement of main memory fetching by instruction buffering and branch anticipation
NMI = number of microinstructions fetched from micromemory
N1A = number of operand 1's fetched from main memory
N1I = number of operand 1's fetched from micromemory
N2A = number of operand 2's fetched from main memory
N2I = number of operand 2's fetched from micromemory
MAT = main memory cycle time
MIT = micromemory cycle time

Because operand 1 is fetched and stored two micromemory cycles later, N1A and N1I are multiplied by two in the AET equation. All microinstructions, which lead to N=NMA+NMI=N1A+N1I, are either in main memory or micromemory. By the structure of the Tucker-Flynn processor, operand 2 need not be fetched each time but as a worst case N=N2A+N2I. Operand 1 is fetched each time, because it is used in the mask operation and output word.

The Tucker-Flynn processor proposes a micromemory ten times faster than main memory. In terms of the AET equation, this would mean MAT/MIT=10. With this ratio, a considerable difference exists between upper and lower bounds, that is 40 vs 4 (for R=1) and 31 vs 4 (for R=MIT/MAT). If NMA/N=0.1, NMI/N=0.9 and (N1A+N2A)/N=(N1I+N2I)/N=0.5 is considered achievable by normal programming means, then with no instruction buffering and branch anticipation 18.4 is obtained, whereas, with instruction buffering and branch anticipation, it only reduces to 17.5 or a 5 percent reduction. If NMA/N=(N1A+N2A)/N=0.1 and NMI/N=(N1I+N2I)/N=0.9 is considered achievable by good data management or interleaving memory modules, a 59 percent reduction can be obtained over the third case without implementing instruction buffering and branch anticipation.

Conclusions

In deriving an equation to represent the average microinstruction execution time for the Tucker-Flynn, dynamically microprogrammed processor, the following points were noted:

1) The microinstruction address register, the accumulator, the general purpose registers, and the data mask registers were assumed to be accessable from high speed registers simultaneously.
2) A single adder can be used for sequencing, effective address calculation, and arithmetic and logical operations.
3) Only one word can be addressed at a time in the memory which consists of both main memory and micromemory.

For the Tucker-Flynn processor, with its micromemory ten times faster than main memory, it has been determined that instruction buffering and branch anticipation will only decrease the average execution time by 5 percent for a particular example. On the other hand, an example representing good data management or interleaving of memory modules can yield a considerably greater reduction of 59 percent.

References

[1] S. S. Husson, Microprogramming Principles and Practices, Prentice-Hall, Englewood Cliffs, N. J., 1970.

[2] M. J. Flynn and R. F. Rosin, "Microprogramming: an introduction and a viewpoint," IEEE Trans. Comput. vol. C-20, pp. 272-731, July 1971.

[3] P. M. Davies, "Readings in microprogramming," IBM Syst. J., vol. 11, pp. 16-40, 1972.

[4] R. W. Cook and M. J. Flynn, "System design of a dynamic microprocessor," IEEE Trans. Comput., vol. C-19, p. 213-222, Mar. 1970.

[5] A. B. Tucker and M. J. Flynn, "Dynamic microprogramming: processor organization and programming," Commun. Assoc. Comput. Mach., vol. 14, pp. 240-250, Apr. 1971.

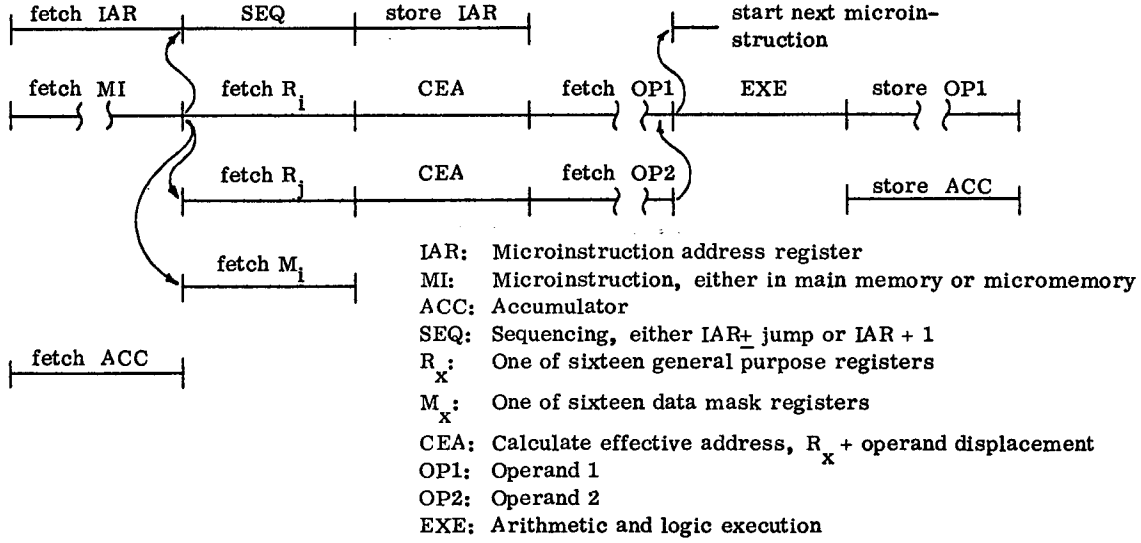

Figure 1. Microinstruction execution sequence.

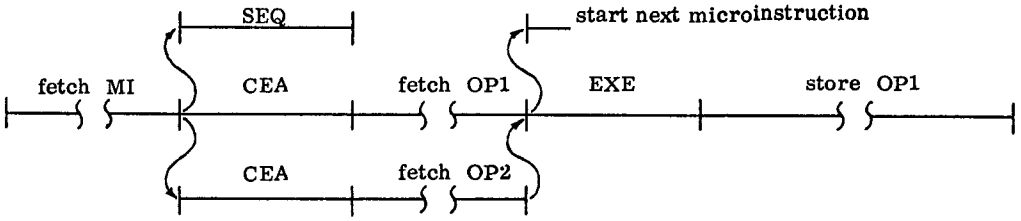

Figure 2. Microinstruction execution sequence with high speed registers.

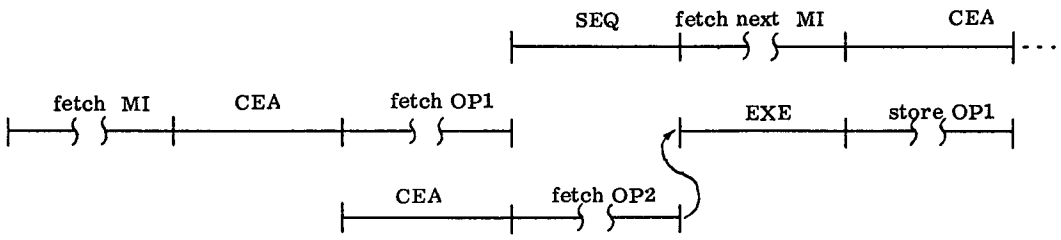

Figure 3. Microinstruction execution sequence with high speed registers, a single adder, and a single port memory.