

IBM CONFIDENTIAL

ADVANCED COMPUTING SYSTEMS
SAN JOSE
February 23, 1966

MEMORANDUM TO:

File

SUBJECT:

DYNAMIC INSTRUCTION
SCHEDULING (DRAFT)

L. Conway
B. Randell
D. P. Rozenberg
D. N. Senzig

001

L. Conway
Archives

DYNAMIC INSTRUCTION SCHEDULING

INTRODUCTION

The order in which the instructions comprising a program are to be executed is normally assumed to be given by the order in which the instructions are held in program storage and by the sequencing control indicated by transfer and conditional transfer instructions. However a programmer, or compiler, can produce many different but equivalent versions of a program merely by making minor alterations to the sequence in which instructions are placed. Normally the actual choice among these alternative sequences will be somewhat arbitrary, though careful programming or compilation often involves an attempt to design a program whose detailed sequences are tailored to make best use of a computer's control and functional capabilities. This can be particularly worthwhile for computers whose internal organization has been designed to attempt to overlap the use of its various functional capabilities.

Take, for example, a computer which initiates execution of instructions in strict sequence, without necessarily awaiting the completion of one instruction before execution of the next instruction, provided that the operands of the second instruction are ready, and the necessary busses and functional units are available. On such a computer the sequence (written here for convenience in a 3-address format)

$$R_1 + R_2 \rightarrow R_3$$

$$R_1 \times R_4 \rightarrow R_5$$

$$R_6 + R_2 \rightarrow R_7$$

$$R_3 \times R_6 \rightarrow R_8$$

might well be preferable to

$$R_1 + R_2 \rightarrow R_3$$

$$R_6 + R_2 \rightarrow R_7$$

$$R_1 \times R_4 \rightarrow R_5$$

$$R_3 \times R_6 \rightarrow R_8$$

if the adder and multiplier were independent functional units.

002

Thus if really effective use is to be made of the internal capabilities of such a computer, careful attention must be paid to the detailed sequencing of instructions in frequently executed portions of a program. This 'scheduling' can be done by an ambitious optimizing compiler, or an extremely conscientious hand-coder. There is often, however, a difficulty in achieving really optimum sequencing by such means--that of the effects of memory interference, which if present will cause variations in the times which operands take to reach the arithmetic and control unit from storage. The effects of such memory interference will not usually be calculable in advance of program execution, particularly if the interference is caused by autonomous I/O units using the memory. Thus there is often cause to consider the possibility of supplementing (or even replacing) the static scheduling performed by coder or compiler by dynamic scheduling performed by the computer as it executes a program. In this paper we describe a technique of dynamic scheduling permitting non-sequential instruction execution. Furthermore, the technique presented is shown to be capable of controlling the simultaneous execution of two or more instructions at a time on machines with sufficiently generous bussing and functional capabilities. In any actual computer design care would of course have to be taken to ensure that any possible gains achieved by such dynamic scheduling were not offset by the cost (both in speed and in circuits) of the extra hardware necessary to perform the scheduling.

The scheme presented uses a very general, but conceptually simple, method of controlling non-sequential instruction execution, and of identifying groups of instructions which are mutually independent and can be executed simultaneously. Brief descriptions of earlier schemes for achieving some of these aims have been given by Amdahl [1], Chen [2], and Thornton [3].

NON-SEQUENTIAL INSTRUCTION EXECUTION

In this section we restrict our attention to the sequencing of straight line coding comprised of instructions, the locations of whose operands and results can be determined directly from the instructions themselves, rather than needing any address computation to be performed.

The sequence in which a series of instructions have been written implies the total effect that these instructions are intended to have when executed. Each separate instruction contributes to this total effect by performing its operations on the contents of certain registers (accumulators, index registers, indicators, etc.) and setting its results into other registers. A dynamic scheduling technique has to insure that any instructions obeyed out of sequence do not change the contents of any registers which are to be used by any instructions whose execution has been delayed temporarily.

A simple set of rules for determining if a given instruction can be obeyed out of sequence is as follows:

- (i) The required busses and functional units are available.
- (ii) The instruction must not use any registers which are used as result registers by instructions whose execution has been initiated but not yet completed.
- (iii) The instruction must not use as result registers any registers which are used as operand registers by any preceding instructions which have not yet been initiated.
- (iv) The instruction must not use any registers (either as result or operand registers) which are used as result registers by any preceding instructions which have not yet been initiated.

These checks can be made in a systematic fashion using what are here called 'sequencing matrices'. Two matrices are used, namely a 'source matrix' (S) and a 'destination matrix' (D). At each cycle, when the machine is attempting to choose an instruction to be executed, rows in these matrices are set up corresponding to each of the instructions which are being considered by the scheduling mechanism. (The cycle referred to above is a clock cycle, which corresponds to the maximum rate at which instructions can be initiated, and will presumably be much shorter than a storage cycle.) The elements in each row of the matrices indicate whether a given register is being used, or will be affected, by the corresponding instruction.

004

L. Conway
Archives