HARVEST REPORT #14

Subject: Control System for a Simple Continuous Stream Register

By: J. H. Pomerene, W. A. Hunt, F. P. Brooks, Jr.

Date: February 1, 1957

February 1, 1957

HARVEST REPORT #14

Subject: Control System for a Simple Continuous Stream Register

Introduction

The continuous stream register is essentially a parallel to serial data converter with provision for automatically maintaining a specified flow of bytes. This report considers a simple streaming mechanism in sufficient detail to show feasibility, checking features, and some of the consequences of interaction with other devices. The treatment is intended to be rigorously asynchronous and self-timed.

The Streaming Concept

The net behavior of a streaming mechanism has been given in some detail in "Machine Specification Report #2" (September 6, 1956) by S. W. Dunwell. Other specification reports describe devices which will work with the streaming mechanism. A complete streaming operation could involve several continuous stream registers together with some logical or arithmetic operation on the bytes of the stream. Figure 1 suggests a typical streaming setup in which data is sent from memory separately into two registers (CSR #1, CSR #2) and the two output byte streams combined into one stream which is assembled back into words and sent back to memory.

It is expected that a stream operation will be defined at the outset by suitable setup instructions followed by an execute or "start" instruction, after which the appropriate memory references and flow of bytes will be sequenced automatically. It is the implementation of this automatic sequencing which is considered here for the case of one CSR accepting successive words from memory and feeding a stream of bytes to some undefined logical operation. Figure 2 shows how this limited problem is abstracted from the general stream process of Figure 1.

The Continuous Stream Register

The CSR itself consists of a 128 bit transistor toggle register subdivided into two 64 bit sections, each of which can independently be filled from memory via the memory data bus. Each of the 128 bits feeds a line of a switch matrix as shown in Figure 3. Eight orthogonal

intersecting lines pick up the byte output and present it to the byte size selector. A maximum size byte of 8 consecutive bits can be selected starting at any of the 128 bit positions by energizing the appropriate byte select line. A byte select line is energized by placing the corresponding 7 bit byte address in the byte address register. In Figure 3, the byte address 59 will energize the select line shown at the center of the matrix and connect bit 59 to the uppermost of the eight horizontal byte lines; bit 60 to the next byte line; etc. The lower 3 byte lines are connected to the first 3 bits of the next word, that is we have here asked for a byte which is partly in the "0" word and partly in the "1" word. The use of a double length CSR keeps this possibility from being too awkward.

The switching elements of the matrix are assumed to be transistors so that, except for an approximately 50 microsecond transient period following imposition of a new byte address, the selected byte output is steadily energized. Should dynamic switching elements such as magnetic films be employed it will be necessary to provide an output byte register.

The Simple Stream

A simple stream operation would begin with the main computer control arriving at a stream setup order. Figure 4 shows an example in which the setup has specified a stream of ten 8-bit bytes to be abstracted from memory starting at bit address 0 in memory word S. The main computer control next encounters a stream "execute" instruction which specified what operations will be done on the abstracted bytes. We will not consider these operations but will use the "execute" instruction only to mark the time at which stream flow can begin. After ten bytes (in this example) have been processed, the stream is turned off and the main computer control allowed to proceed to the next instruction.

Asynchronous Operation

The preceding paragraphs have presented the streaming operation from the user's standpoint. It has been easy to describe primarily because streaming has been so defined as to make programming easier and more natural. But the automatic features which simplify programming demand more of the equipment than has been usual in the past. Moreover the CSR of Figure 2 is only one component in a computing complex which is at least as inclusive as Figure 1 and may even contain other complete computers. A variety of interactions can occur between the components of this complex. For example two CSR's may refer to the same memory block at the same time. One may proceed, the other must wait. These interactions seem to force a flexible policy of step

by step, or asynchronous, timing of all processes.

Asynchronous timing may be formally defined as follows: consider that the numbers in all the machine registers as well as the contents (0 or 1) of all control flip-flops are assembled together into one large n bit number. Call this number the present "state" of the machine. If m bits of the present state are combined in some way to produce K new bits, a potential new state is defined. The transition from the old state to the new state is complete when these K new bits have been stored correctly in K of the n flip-flops. None of these K flip-flops can be the same as the m which entered into forming K. Therefore at most n/2 bits can enter into formation of the next state. These rules guarantee that the new state can be formed, stored, and checked - however long this may take - before the relevant portion of the present state is destroyed.

Asynchronous operation is illustrated throughout the following sections. A specific example is discussed in more detail in the section on address generation.

Overall Timing Control

The whole stream operation is first divided into operations per byte. Each per-byte operation is further sub-divided into a control phase followed by a data phase. In the control phase is done everything necessary to insure that a correct next byte is presented at the CSR output. In the data phase the byte remains presented at the CSR output until it is no longer needed by the logical process being performed.

The timing control at this level is shown in Figure 5. Starting at the upper left the receipt of a stream setup instruction sets the C/D toggle (flip-flop) to the control (C) state and starts the control phase. This may proceed even though the execute order has not yet been received. After all requirements of the control phase have been satisfied, the "control phase done" line is turned on. When an "execute" order has also been received, the C/D toggle is set to the data (D) state and the data phase begun. After the data phase is over, C/D is set back to C to prepare the next byte.

A byte counter, lower right, advances by one for each alternation of C and D phases. The exact duration of either C or D phases does not matter at all to the operation of this counter provided a given phase lasts at least long enough to allow a correct transfer between n^1 and n_1. This minimum duration will be guaranteed by not allowing the C$\longrightarrow$D transition until a check circuit (not shown on fig 5) signals $n^1 = n_1$.

A simple but important property of the asynchronous approach has already appeared. We have broken the overall timing problem down into two independent subdivisions without placing any restrictions on the subdivision timing whatever except to point out that they must last long enough to do what they may be required to do (but no longer). This property can extend through all levels of the design. The practical importance of this concept cannot be overstressed. It means that a design meeting logical and mathematical requirements can be laid out before specific component performance is known, and further that at any later time a faster component can be used without invalidating that design.

Control Phase Timing

Receipt of a setup instruction starts the control phase preparatory to emitting the first byte. Referring to Figure 6, lower left, the setup condition gates the starting byte address S_B into the "next byte address" register and, because we are in the C phase, also into the "present byte address" register. This latter register is the same as that shown in Figure 3, hence the switch matrix sets up the proper byte location although the CSR may not yet contain the proper word. The lowest order bit of the word address is included as the 7th bit of the byte address and will select the half CSR in which the byte begins.

A toggle associated with each CSR half (described later in connection with Figure 9) denotes whether that half has been filled with the proper word. The state of bit 7 of the byte address is compared with the corresponding CSR full/empty signal, or both if the byte overlaps 2 words. If the register(s) are filled and all other C functions are complete, the C/D toggle is switched to the D phase. An analogous data phase control is indicated but not discussed.

Memory Processes

The preceding section describes the control phase fully, assuming the interaction of the full/empty toggles. The behavior of these is a function of the memory reference process which comprises the remainder of this report.

Memory Address Generation

Figure 7 shows that at setup time the starting memory address S is placed in the memory address register. The initial conditions are such that the gate $S + 1 \rightarrow S'$ is enabled and therefor S + 1 will immediately start forming into S'. A feedback check on this gating process

is provided by forming S' - 1 (which should be $\boxed{(S + 1) - 1} = S$) and comparing the result with S. An equality condition signals correct completion of the gate.

When the initial memory address is no longer needed, that is when the first word has been brought to the CSR and checked to be correct, gate $S + 1 \rightarrow S'$ is shut off and gate $S' \rightarrow S$ is enabled, bringing in S + 1 as the next memory address, etc. Completion and check of the $S' \rightarrow S$ gating is signalled by a digit for digit comparison of S and S'.

The control of this asynchronous process is examined in more detail in Figure 8 (a) and (b). As described earlier, the initial state of this system is given by the two 20-bit numbers S and S' (S' initially not defined) or more precisely by the 40 bit number SS'. Next a new state S(S + 1) is formed and checked. Then the state (S + 1)(S + 1) is formed and checked etc. The 4 successive steps of this process are summarized in the state table (Figure 8a) and a two bit binary code used to identify each step. Notice that a reflected or "Gray" code is used because each successive step should be represented by only a one bit change. The table implies the gating control block diagram of Figure 8b. The two control toggles are necessary to define the 4 successive steps. For each "get next memory address" command one progression is made through the 4 steps and an advance of one made in the memory address.

Memory Reference Control

Figure 9 shows the control for initiating and checking memory references. Starting at the upper left and assuming that we want a memory reference to address S, we transmit the address S over the data bus to the memory. Along with S is sent the identification code of the unit (CSR in this case) which is to receive word S when it comes out. Assuming 2.0 microseconds memory, the 7 high order bits of S interact with recognition circuits in the memory boxes and select a particular box. The remaining 13 bits plus the receiver identification are, on recognition, gated into a 17 bit address register in the memory box. Receipt of a selecting address initiates a memory cycle in the selected box. The particular wire actually pulsed in the memory matrix is checked against the address in the register (ref. a scheme proposed by W. Hunt). When the selected data is available it is transmitted over the data bus back to the CSR control. Since the selecting address may have reached the memory incorrectly for a number of reasons, the address actually used is transmitted back, along with the receiver identification, to the CSR control. Here it is checked against the address actually sent.

The possibility of error due to logical interference between various users of the memory is considerable and so a properly requested memory reference may fail to compare on the address echo check. The whole stream operation will consequently wait, so it will be advisable to incorporate a "try again" circuit (not shown) on the memory request line. Some provision in this direction will be needed anyway since the requested m emory block may be busy. Once an echo check is obtained, it is routed toward the proper CSR half (in Figure 9 the control peculiar to the 0 half CSR is omitted for simplicity). The major logical flow in Figure 9 is drawn heavy for emphasis.

Assume the 1 half CSR is designated, and if it is indeed empty, the memory word is gated into it. The correctness of the data itself is checked in the CSR by at least a parity check and possibly a Hamming type check. However it is done, a check signal is or is not emitted. If the check fails the aforementioned "try again" circuit comes into play. If the check goes, the full/empty toggle is turned to "full". The "full" condition allows the next memory address to be set up and, provided the other CSR half is now empty, initiates another memory request.

Memory Busses

A partial aim of this analysis is to present the demands of the CSR mechanism in sufficient detail to be of use to the memory and the memory bus designer and yet by adherence to asynchronous techniques to make the present proposal independent of how the memory problems are finally met.

It should be noted that since the memory busses may be electrically quite long and since memory references form a closed loop with respect to the requesting unit, the proposed checking scheme has been an overall one. An important implication of this choice is that the data transmitted over the busses can be a burst only long enough to pass a comparing circuit and set a register (say 70 millimicroseconds) however long the busses may be.

J H Pomerene

JHP/jh

J. H. Pomerene

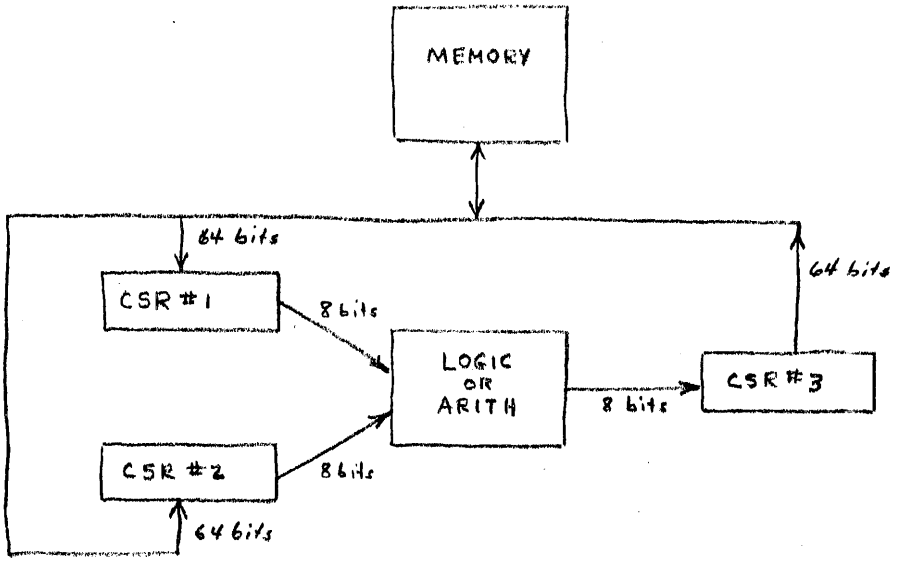


Fig 1

DATA FLOW FOR A STREAM PROCESS

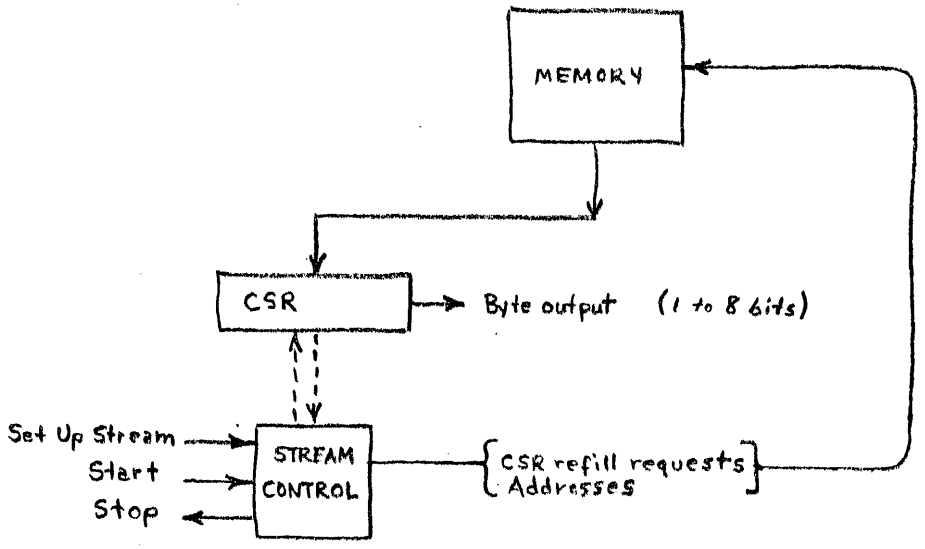


Fig 2

SIMPLE STREAM CONTROL

23 Jan 57
Pomerene

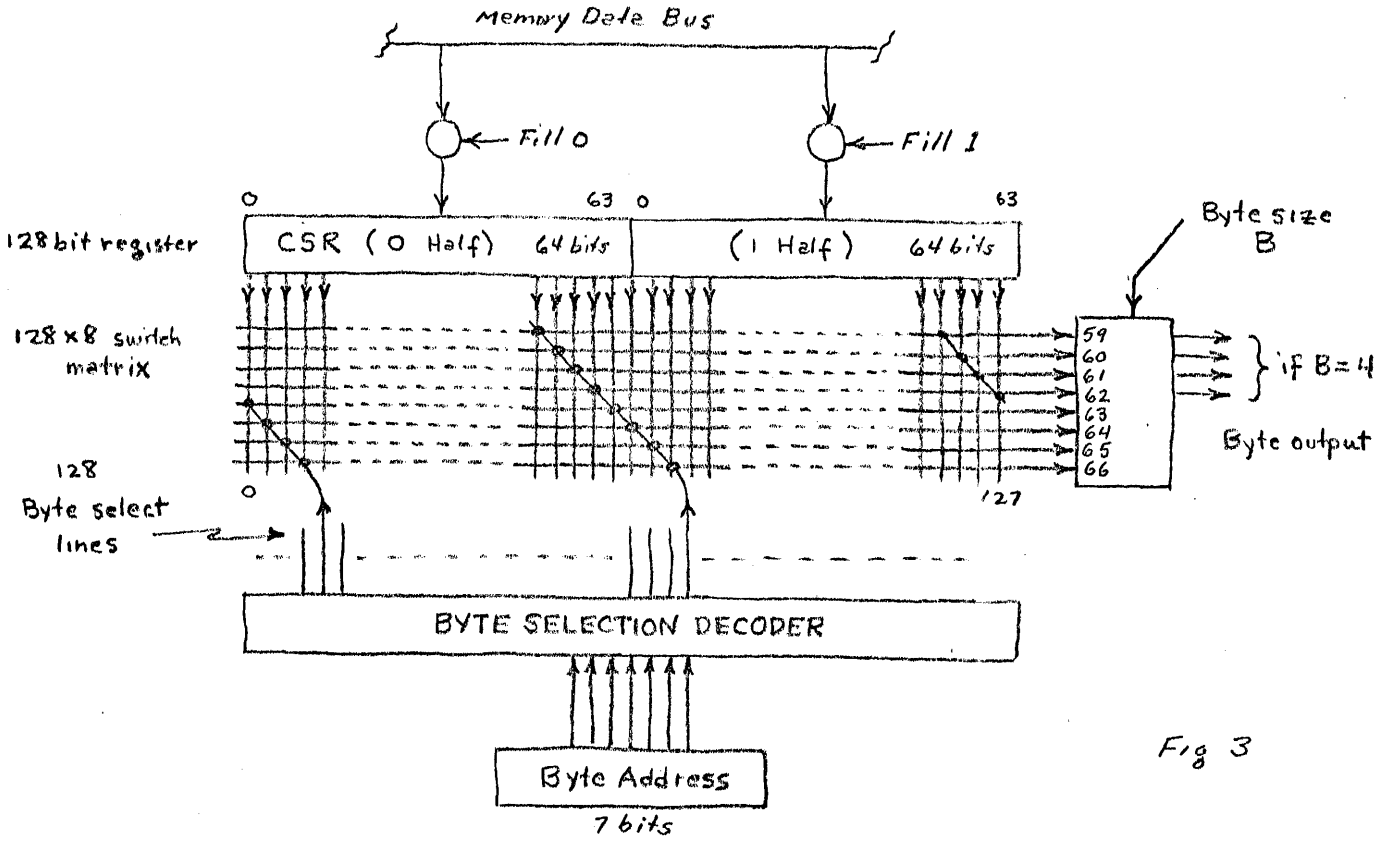


Fig 3

DETAIL OF CSR SHOWING BYTE SELECTION MECHANISM

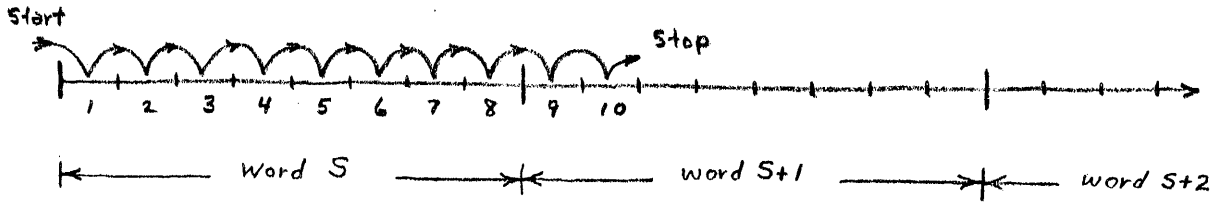
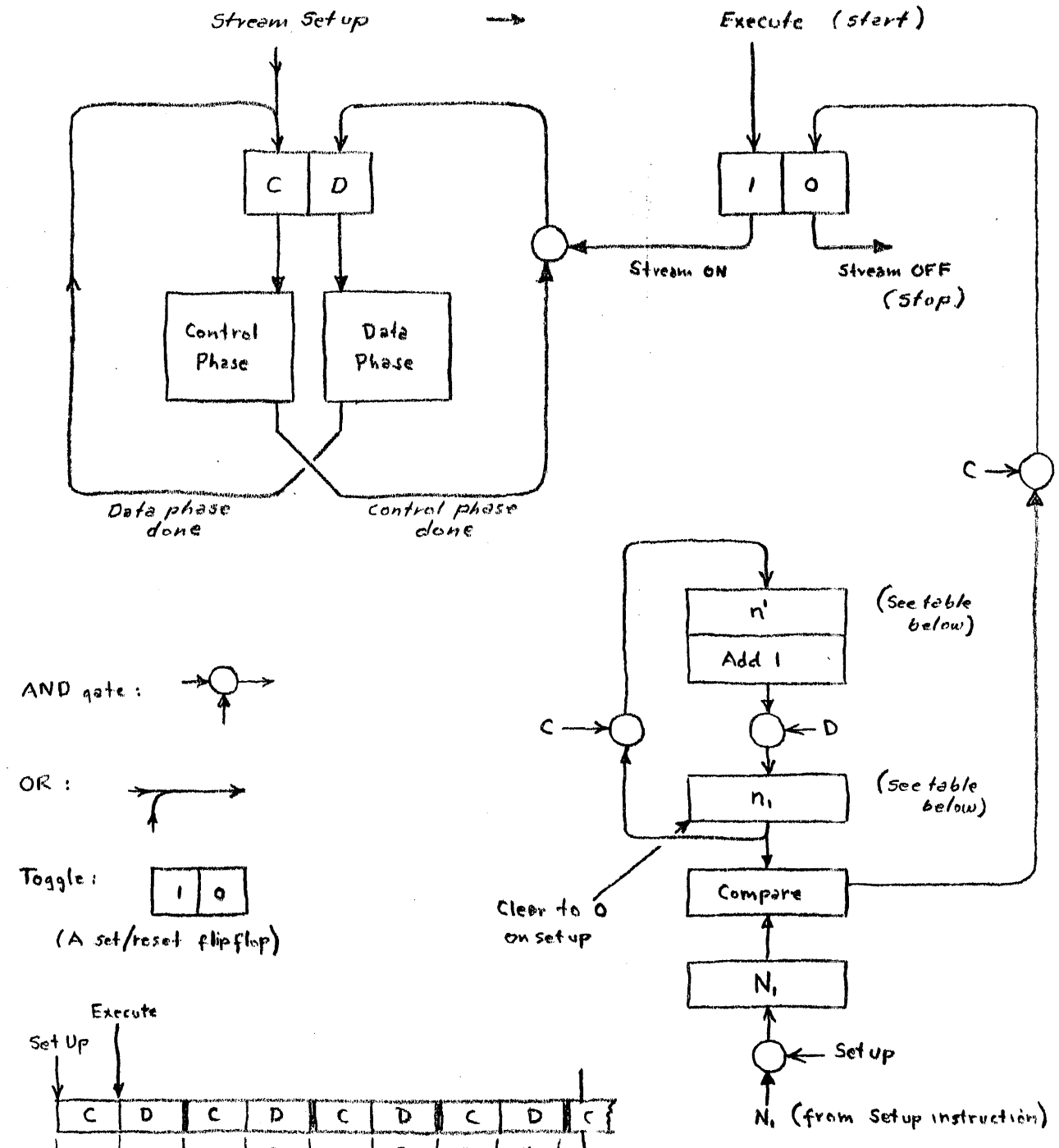


Fig 4

Diagram of streaming operation
starting with word S in memory
successive 8 bit bytes from
bit address 0
Stream of 10 bytes ($N_1 = 10$)

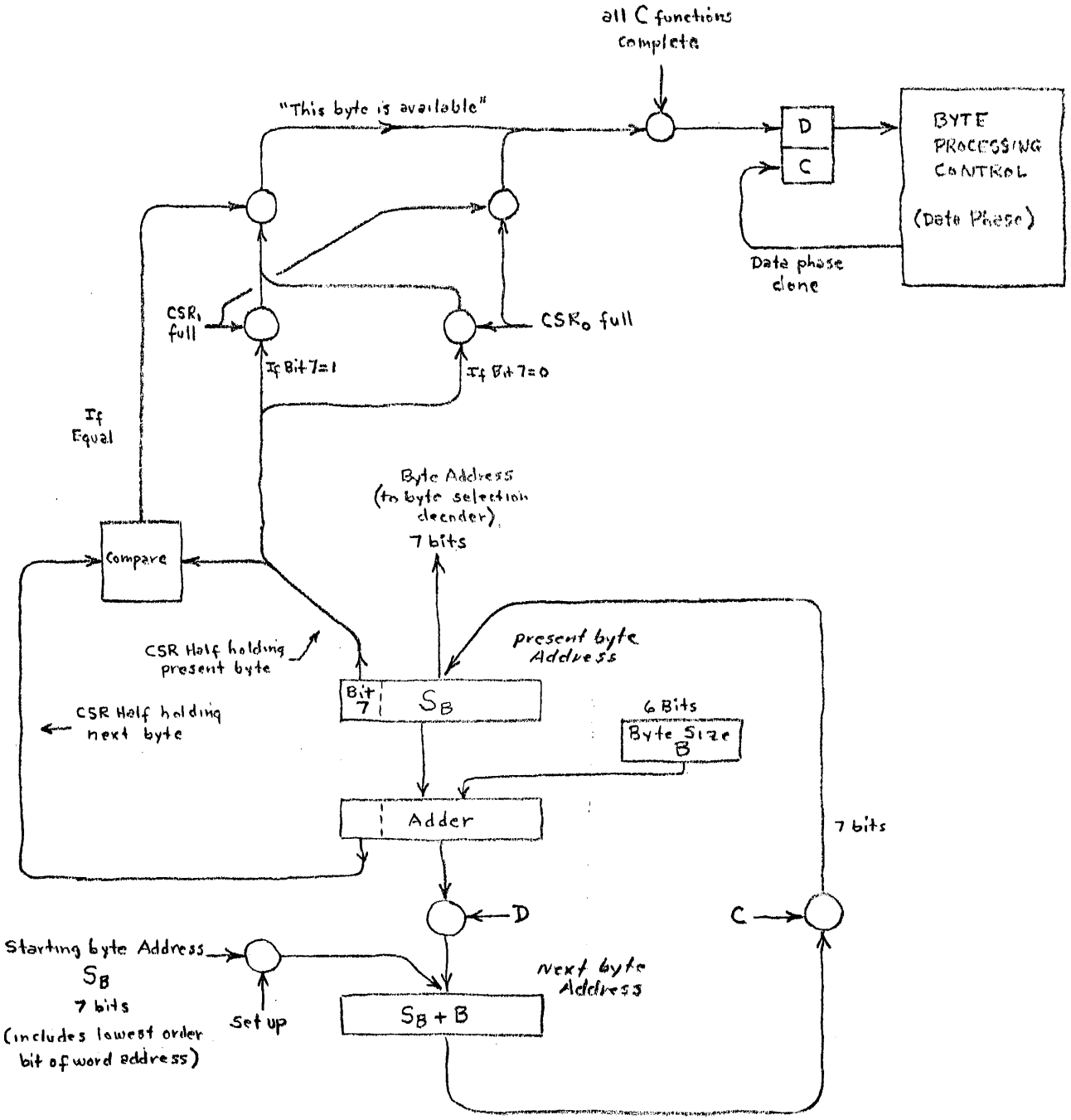
23 Jan 57
HRonerene



Timing example for $N_1 = 4$

OVERALL TIMING CONTROL
Simple CSR
21 Jan 57 JHaanstra

Fig 5



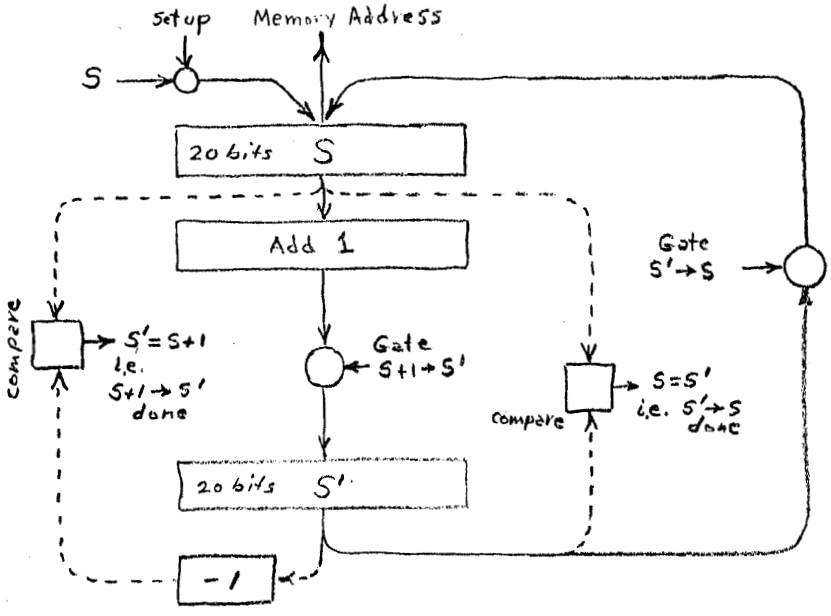
BYTE ADDRESS GENERATION

AND

CONTROL PHASE / DATA PHASE TIMING

Fig. 6

23 Jan 57
HBomerene



Step ① (while S is being used as a memory address) start forming S+1 and gating it into the "next address" storage register S'.

② Check and record that S' has received S+1 correctly.

③ Start gating S' into S for use as the new memory address.

④ Check and record that S has received S' correctly.

① As above, etc.

Fig. 7 MEMORY ADDRESS GENERATION

STEP	3	4	1	2
GATE "ON"	S'→S		S+1→S'	
GATE STATUS	in process	Done	in process	done
BINARY CODE	00	01	11	10

Fig 8a
State Table

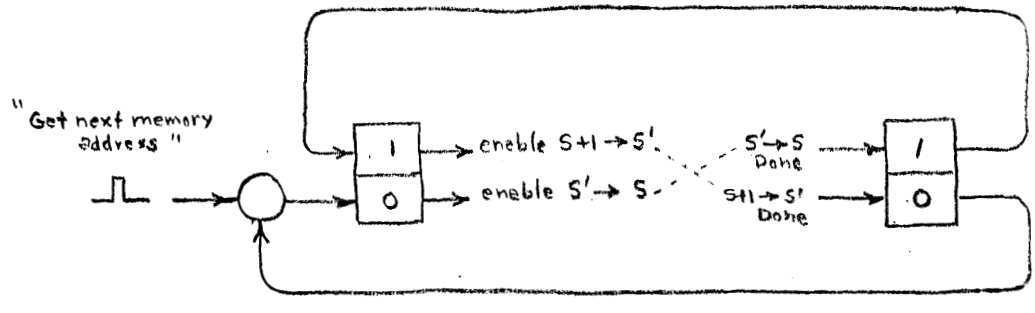


Fig. 8b

CONTROL OF MEMORY ADDRESS GENERATION

22 Jan 57
JPomerene

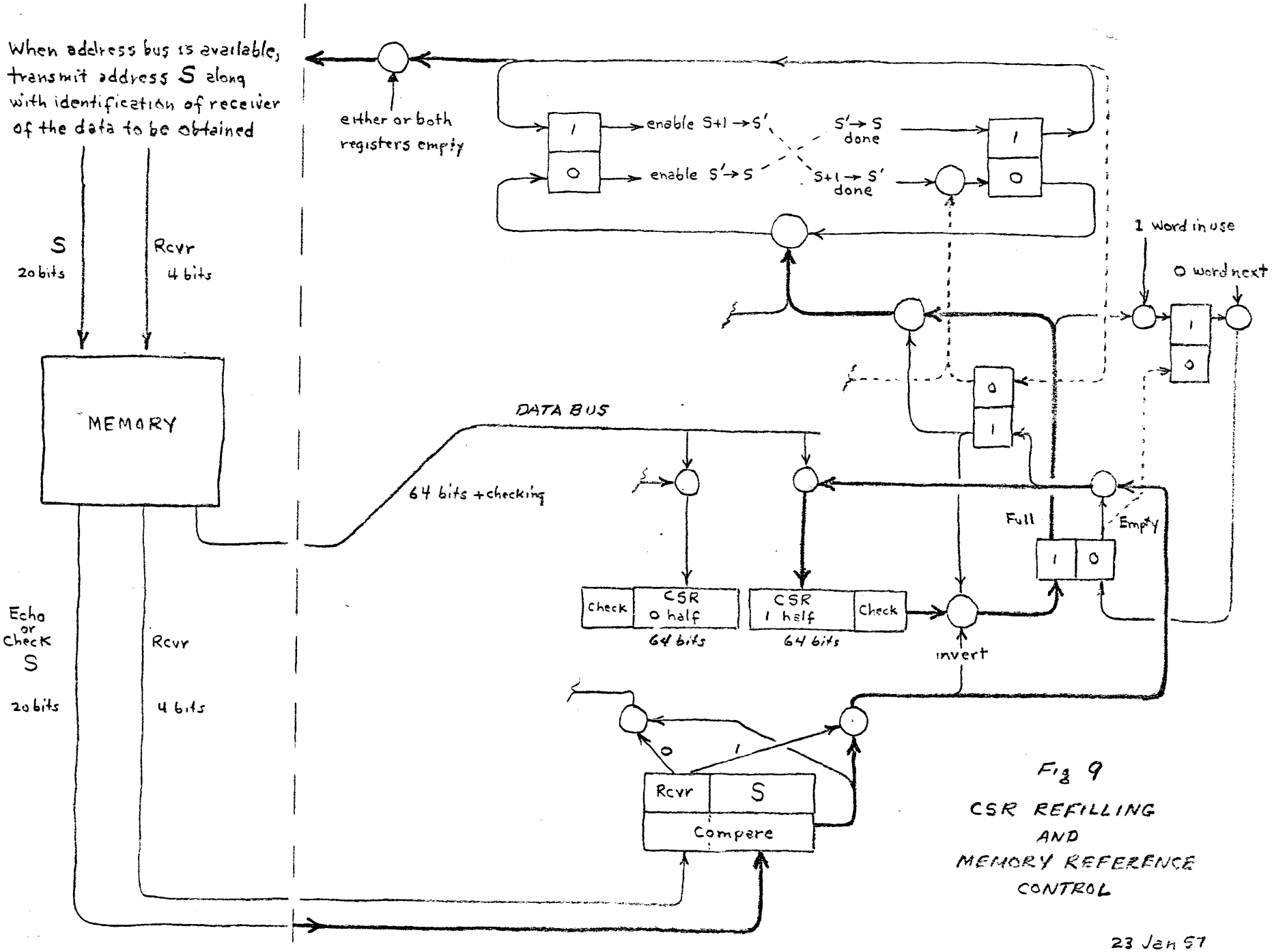


Fig 9

CSR REFILLING AND MEMORY REFERENCE CONTROL

23 Jan 57
JPomerene