Instrumentation Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts

Digital Development Memo #286

To: Eldon Hall
From: Herb Thaler
Subject: Some Idiosyncrasies of Block 2 AGC Logic
Date: December 28, 1965

Arithmetic Unit

The Block 2 AGC arithmetic unit consists of several simple flip-flop registers (A, L, Q, etc.) and a binary adder. The flip-flop registers are much like their Block 1 counterparts, and information transfers between them are handled in a similar fashion. The registers are driven by three clocked action pulses - read, write and clear. If one divides the register transfer interval (1 microsecond) into four successive phases, then the three action pulses can be described as follows:

read — phases 2, 3 and 4 - places the current value of the register being interrogated on a set of data buses (write lines).

clear — phase 3 - the register being set to a new value must first be cleared of its previous contents, else the two values will combine.

write — phases 3 and 4 - samples the value of some data source and gates it into the proper register. Editing of data is achieved by sampling in a fashion other than directly bit by bit. An example of editing on writing is a shift of data by one or more positions as in the AGC shift and cycle registers.

The adder, on the other hand, is significantly different from the Block 1 adder. One of the tricks of logic design which greatly improves the operational speed of such a parallel binary adder has been incorporated. The technique used is called carry skip.

Each stage of the basic adder has three inputs (X, Y, carry in) and two outputs (Sum, carry out). The two input operands (X, Y) are by themselves not sufficient to fully define the sum and carry output - one must also know the value of the "carry in." Unfortunately, one does not determine this

"carry in" value instantaneously without the expenditure of literally dozens of extra gates. Normally this information is provided by one of the adjacent adder stages as a byproduct of its sum computation. That state, too, must be provided with "carry in" information from an earlier stage, and so on down the length of the adder. Thus a string of carry propagation gates is formed by linking successive adder stages. The time delay between "carry in" being inserted into say stage 1 and being propagated through stage 16 for a worst case operand pair can be as high as 800 nanoseconds with the logic circuits currently being used.

In order to circumvent this problem the adder carry chain has "skip gates" placed across it. These gates examine four successive sets of operands (X, Y) and the "carry in" presented to the first of the string. The worst case delay situation cited above is that in which every adder stage propagates a carry applied to its input through to the next stage, but does not generate one by itself. This situation can be determined by examining the four sets of instantaneously available operands, and, if this condition pertains, permit the carry entering the first stage to immediately pop out of the fourth. This process is repeated for various groups of four adjacent bits, resulting in an adder whose carry structure is as shown in Figure 1. The carry propagate and skip paths are joined by logical OR connections, so that the first one to arrive at a junction can immediately go through it. Thus, for example, a carry inserted into stage 1 may reach stage 13 by several paths, the quickest of which will predominate. These path delays range from 600 nanoseconds via all propagate stages to 150 nanoseconds via all skip gates. The worst case carry propagation time in the adder is thus reduced from 800 nanoseconds to 350 nanoseconds (3 skips and 4 propagates).

The Mysterious WHOMP

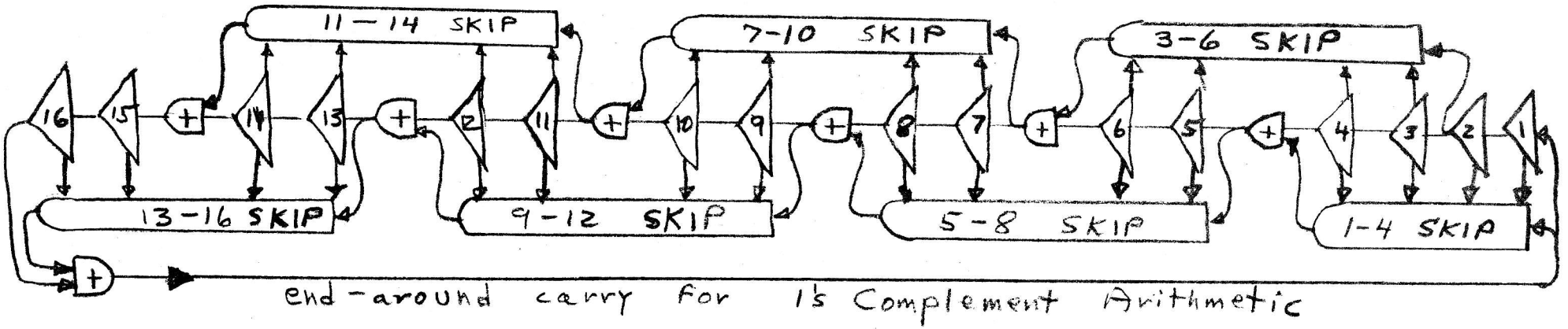
Full use was made of this high speed adder in designing the Block 2 divide instruction. In that instruction, the following three microsecond sequence of events occurs 14 times:

μsec 1 The two operands (X, Y) are entered into the adder.

μsec 2 The adder Sum output is tested and if positive, the X operand is cleared out (pulse CLXC). If the Sum output is negative the

Figure 1

Functional Representation of Block 2 Adder



$$\text{Normal Carry Out} = (X_J \cdot Y_J) + (\text{Carry In}_J) \cdot (X_J + Y_J)$$

$$\text{Carry Skip} = (\text{Carry In}_J) \cdot \left((X_J + Y_J) \cdot (X_{J+1} + Y_{J+1}) \cdot (X_{J+2} + Y_{J+2}) \cdot (X_{J+3} + Y_{J+3}) \right)$$

$$\text{Sum} = (\text{Carry In}) \cdot (X \cdot Y + \bar{X} \cdot \bar{Y}) + (\overline{\text{Carry In}}) \cdot (X \cdot \bar{Y} + \bar{X} \cdot Y)$$

μsec 2 (cont.) X operand is left undisturbed, but a quotient bit represented by pulse RB1F is entered into another flip-flop storage register.

μsec 3 The Sum is read into buffer storage to prepare for the next add-test sequence.

If one looks very carefully at the fine timing structure of this sequence, the following facts emerge. The first addition (μsec 1) is permitted only 750 nanoseconds before the sign test in μsec 2 is performed; and if CLXC is generated the second addition (and it is truly a second complete addition) is granted only 1000 nanoseconds minus the delay of the data bus structure (typically 250 nanoseconds). Thus both additions are only permitted 750 nanoseconds for maximum carry propagate time.

The mechanics of the first addition follow the adder discussion in the previous section, so let us now consider the worst case of the second addition that occurs with generation of CLXC and the subsequent clearing of the X operand. If one operand entering into a sum computation is identically zero, the sum equals the other operand, and the "carry out" levels should be identically zero. This is the desired adder output following a CLXC pulse - (Sum = Y, carry chain all zeros). If the Y operand is all ones and the initial X operand is a low order bit one, then the adder state at the instant the X bit is cleared has carry out at all stages with all skip gates also activated. The carry propagate and skip equations (Figure 1) are such that loss of the single X bit has no effect on the propagate chain once end-around lockup occurs. Thus the adder fails to produce the proper second sum output. To avoid this lockup phenomenon, the WHOMP signal (derived from CLXC) instantaneously zeros selected points in the propagate and skip loops, such that the carry loop is broken. The points selected form an optimum set to minimize carry-loss delay at a reasonable circuit cost.

Memory Cycles and Quarter Codes

The Block 2 memory cycle structure differs considerably from that of Block 1. The basic difference is in the phasing relationship between core rope memory activity and the computer time pulse generator (T01 - T12).

In the Block 1 machine, the twelve microsecond interval beginning with Time 1 and ending after Time 12 was defined as a memory cycle time. Both

erasable and rope memory cycles began with an address determination at Time 1, followed immediately by activity in the proper memory. The proper memory location was always selected in this manner so that memory cycles, once begun, were always carried to completion.

The rope memory addressing technique involves something called strand selection. This requires the establishment of forward current in a pair of back-to-back diodes to couple the core rope output signal through a transformer into a DC coupled sense amplifier. The basic circuit is shown in Figure 2. If the diode forward voltages (V_{d1} and V_{d2}) are unequal, an instantaneous DC bias is introduced into the sense amplifier. This bias diminishes with

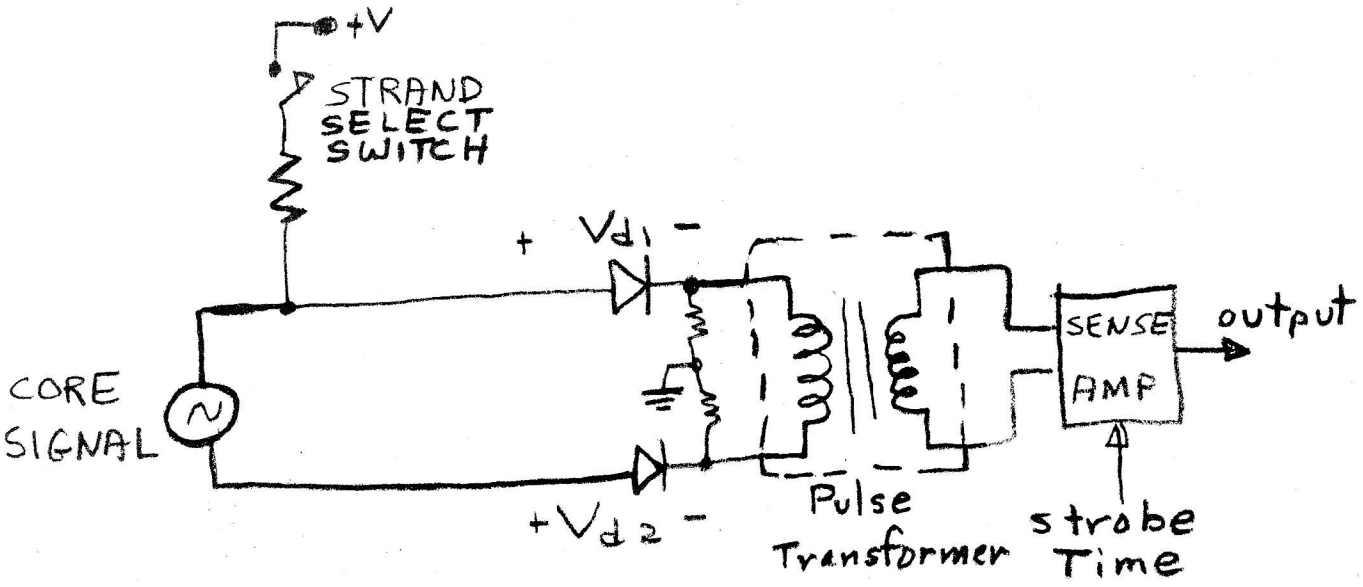


Figure 2

time, since the coupling transformer's low frequency response is limited. Unfortunately the time scales of core signal envelope and memory access time are comparable, so the DC bias introduced in this manner has not completely disappeared at "strobe time." If the strand selection could be made several microseconds earlier, the bias left at strobe time would be substantially reduced.

This has been done in Block 2 by establishing a groundrule that address information shall be available at Time 8. If the address falls into the domain of rope addresses (2000_8 - 7777_8), the cycle is to begin immediately (Time 9).

Since access to the information (Strobe Time) is not required for ten microseconds (the following Time 7), strand selection transients are allowed to decay an order of magnitude further than in Block 1. If the address falls into the domain of erasable memory (0 - 1777_8) the memory activity is postponed until the next Time 2, just as in Block 1.

Note that in Block 2 the field of erasable memory overlaps that of Special and Central memory (0 - 7), causing erasable memory currents to flow when S&C access is made. This has been done to smooth out the current drain on the memory power supply in the interest of reducing memory voltage variations. Although memory currents flow, no strobe is allowed for S&C core locations. To further stress this concept, there is some memory activity during every computer subinstruction except MP1. Activity is initiated in the rope memory during the middle cycles of the divide instruction by setting bit 12 of the address register late in the first divide cycle. This action forces the computer address into the rope memory region after the initial erasable memory data access. Since rope memory can be safely cycled repeatedly without data restoration, no information is lost by this artifice.

Quarter-Codes and Memory Timing

The AGC instruction set is greatly expanded in Block 2 by the use of two techniques - true extension and order code splitting (quarter codes). True extension is achieved by setting a flip-flop with the instruction EXTEND. This flip-flop then specifies that the next instruction in the program sequence is to be interpreted as an extended order code. Thus the AGC three-bit order code is effectively doubled in range by EXTEND.

Consider now the set of 16 available instructions. Some of them such as Clear and Add or Add would be applicable to all memory types - fixed, erasable core and flip-flop. Others such as Transfer to Storage or Exchange could not be sensibly directed to fixed memory, since the nature of fixed memory precludes the modification of its contents by program. In the latter case, one might consider that the order code was therefore applicable only to the erasable memory portion of machine addresses. Since this is only 1/4 of available addresses, it is obviously wasteful of two bits of instruction memory. The origin of the quarter code concept lies in the desire to utilize this wasted information for additional instruction specifications.

Each of the original 16 order codes which could be thus restricted is subdivided into four quarter codes. These are distinguishable by bits 11 and 12 of the instruction word. Take the case of INDEX as an example. Order code 5 and address 0 - 1777 is interpreted as INDEX (0 - 1777); code 5 and address 2000 - 3777 as DXCH (0 - 1777), 5 and 4000 - 5777 as TS (0 - 1777); and code 5 address 6000 - 7777 as XCH (0 - 1777). Note that although the original address may have been 2000 or greater, the final interpreted address for all four quarter codes is in the same erasable memory field (0 - 1777), but the instruction executed depends on the initial address range. Note also that the original address is placed into the address register at Time 8 as previously described. Since the last three quarter codes actually lie in the fixed memory address field, fixed memory cycles are initiated to the location specified by the initial address. Each of the quarter code subinstructions is responsible for clearing S-bits 11 and 12 at Time 1 by control pulses RL10BB-WS to achieve the restriction to erasable memory. Meanwhile the fixed memory has undergone half of its own cycle, and must be restored to its initial rest condition. This is done by a rope clear current which is energized whenever a rope address is present at Time 10 but not at the following Time 2.

What about Erasable?

Just as the rope memory must be restored to a rest condition, the erasable memory must also be left in its natural state following an E-Memory cycle. In this case, the natural state is with the proper information restored into the memory location just accessed. As previously mentioned the E-Memory cycle starts at Time 1, just as in Block I, but the address register is allowed to change at Time 8. This is not enough time to permit the necessary data restoration (see Figure 3).

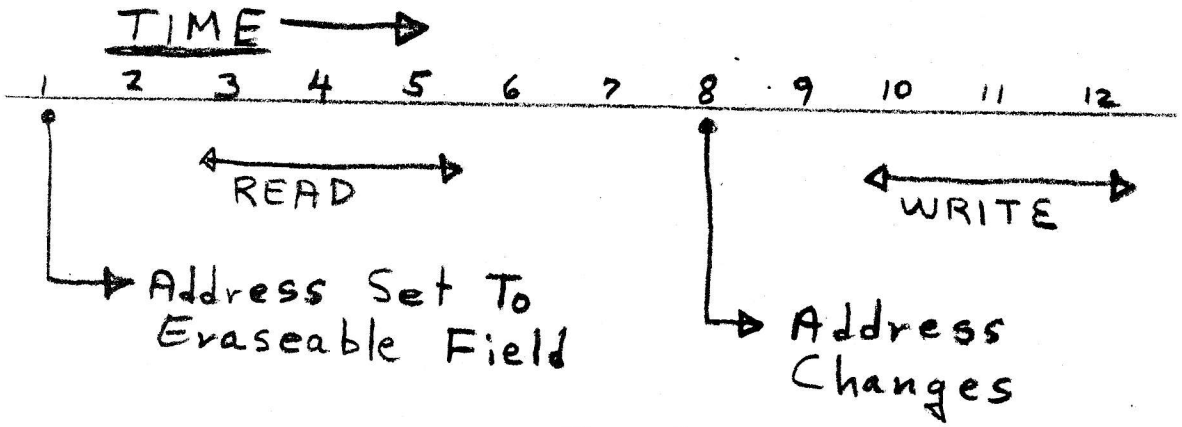


Figure 3

The original address set up at Time 1 must be remembered by the Erasable Memory Switch Core array to effect writing back into the proper location. Also the memory timing logic must remember that an Erasable Memory cycle was begun, so that it will be properly finished.