F-3 THEORY OF OPERATION

Overview & Main Data Paths

The data paths of the F-3 are, except where noted, 36 bits wide. The central part of the data paths are nine 2901 IC's (Integrated Circuits), which contain an array of 16 registers, an ALU (Arithmetic-Logic Unit), a temporary register called Q, and some control logic. The 2901 has a data output which is called the OBUS, and which is the main data bus in the F-3 from which registers get loaded. The 2901 also has an external data input. In the F-3 the 16 registers are used as the AC's (accumulators).

The ALU in the 2901 is capable of taking two words and performing one of eight operations on them, the result of which is (usually) placed on the OBUS, and may also be loaded into one of the 16 AC registers or the Q register. (In addition, the result may be shifted left or right by one as it is stored into a register. Also, the Q register can be caused to shift by one.) The eight operations are: add, subtract, subtract-in-the-other-order, OR, AND, XOR, equivalence, and AND-with-one-operand-inverted. The two operands may come from the AC registers, the Q register, or the data input; or one of them may be 0. So, for example, one can add the contents of the Q register to the data on the data input, and put the result back into Q, or into one of the AC's. The result could also be loaded into one or more of the external registers connected to the OBUS.

The data input of the 2901 can come from any of nine sources (detailed below). The selected source can be rotated and masked before going into the 2901 data input. The rotation is a 36-bit left rotate of any amount from 0 to 35 bits. The amount is specified in the micro-code, or the micrc-code can specify that the rotate amount should come from a special register for that purpose. After the word has been rotated, it can be masked. Masking consists of ANDing the word with a mask word which has M low order bits on (1) anc 36-M high order bits off (0). M, the mask-size, can be any number from 0 to 36, and is specified in the micro-code; or the micrc-code can specify that the mask size should come from a special register for that purpose.

One of the nine sources of data is an array of 256 words (36-bits each) called the A-MEM. The other eight sources are: the AR register (see below), memory data (see under memory-access), the mask word, the constant word (see below), a wcrd with the PC register (macro Program Counter register) in the right half and CRYOR (flag and status) register in the left half, the MA (Memory Address) register (right half only), I/O data, and the IR (macro Instruction Register).

If the mask word or the constant word is selected, masking does not happen. Consequently, a mask word of, say, seven bits can be selected, and then rotated by six bits, and the resulting 2901 input would be 17700 (octal). The constant is also unmasked, and can be any six-bit number, or the number 1,,1 (1 in low-order bit of each half-word). The constant can also be rotated.

There are a number of external registers, each of which has a special purpose. The AR is a 36-bit register which can be loaded from the OBUS on any cycle, and which has several special purposes. There is a comparator which compares the magnitude of the high-order 6 bits of the AR with the next lower 6 bits. Also, the AR (low order 16 bits) holds the micro-memory address when storing into micro-memory.

? what comparator?

The PC is an 18-bit register which is also a counter. It can be caused to increment by 1 on certain cycles.

Two registers which are involved in main-memory access are the MA and HOLD. In general, the MA (which is 18 bits wide) has the memory-address of the location being referenced, and the HOLD (which is 36-bits wide) holds the data being stored on writes. Note that the MA may be loaded either from the OBUS, or directly from the PC. Memory access is covered in more detail later.

The IR is a 36-bit register which is intended to hold the macro-instruction being executed. Its left half, which has some decoding logic attached, is loaded directly from the in-coming memory data, rather than from the OBUS (to save time); while the right half is loaded from the right half of the OBUS. The IR may be loaded in parts. The options are: load the whole IR (36 bits), load bits 13 thru 35 (23 bits), or load only bits 18-35 (18 bits).

The flag, or CRYOV, register is an 18-bit register loaded from the left half of the OBUS, except that bit 10 and bits 13 thru 17 are missing. Several of its bits have special purposes.

The ROT SIZE register is a six-bit register loaded from the low-order 6 bits of the OBUS, which, under micro-code control, can supply the rotate amount (0 to 35 bit left rotate).

The MASK SIZE register is a six-bit register loaded from the low-order 6 bits of the OBUS, which, under micro-code control, can supply the mask size (0 to 36 bits, see above).

The AC-SEL register is a four-bit register loaded from the low-order 4 bits of the OBUS, which, under micro-code control, can supply the number (address) of the AC (in the 2901) to be accessed. The AC-SEL register can also be used to supply the low-order 4 address bits for the A-MEM (see below) and also can be caused to increment by one under micro-code control.

The DEV-ADR register is a five-bit register loaded from the low-order 5 bits of the OBUS, which holds (usually) the number of the I/C device being talked to. It also can supply the high-order 5 bits of A-MEM address.

The INT-ADR (interrupt address) register is a 5 bit register loaded from the INT-ADR lines. It holds the number of the interrupting device, and may be used to drive the DEV-ADR lines ano adcress the A-MEM.

The MAP-DISP register is covered in the map section.

The IOD register holds data which is to be sent to an I/O device during a normal (non-DMA) I/O transfer.

The HI-ABS-MA is used to supply high-order memory address bits when the map is not in use. It is loaded from bits 14-17 of the OBUS, and is discussed in more detail in the map section.

The MAP-EXEC-SR is discussed in the map section.

Main Memory Access

Every time the MA is loaded, the main memory does a read cycle on the following cycle, and the data from that read will be available at the begining of the cycle following the read cycle. The cata is accessed by selecting memory data as the external source. The only exceptions to the read cycle are: if STRT-WRT is specified as an additional destination, a write cycle is done instead of a read; and if the map is turned on, but not valid for the MA address, the cycle is prevented. Note that the MA may be loaded either from the OBUS cr directly from the PC (while the OBUS is being used for something else).

Data to be stored on writes is usually taken from the HOLD register (however data stored on writes may also be taken from the I/O bus). There are three destination designations relevant for writing: STRT-WRT, HOLD, and MEM-STO. Destination MEM-STO causes the HOLD register to be loaded from the OEUS, and a main memory write cycle to be taken on the following cycle. HOLC causes the HOLD register to be loaded, but no cycle to be taken. STRT-WRT causes a write cycle to be taken on the next cycle, but the HOLD register is unchangec.

Whenever the HOLC register is loaded, (by destination HOLD or MEM-STO) its cutput is selected as the memory data which is seen whenever the memory data is selected for the external data source. This condition persists until the next time the MA is loadec, when the selector goes back to looking at the data coming in from main memory. This means that any time a write is done followed by a read, the data will come from the HOLD recister; until the MA is loaded with a new adcress and a new read cycle happens (now reading data from the main memory instead of the from the HOLD register). Also, it means that data generated in the 2901 can pretend it is coming in from main memory, by being placed in the HOLD register (very hancy for accessing AC's as if they were main memory). Additionally, the HOLD register can be used for temporary storage.

MAP

For mapping purposes, all memory addresses are divided into two parts. The low-order nine bits do not get mapped and are sent directly to memory. If the map is enabled, the high order nine bits, refered to as the virtual page number, get mapped by substitution of a new, or physical, page number; which is obtained by table look-up. The process consists of simply indexing into the map table by the virtual page number to acquire the physical page number (which may be up to 13 bits long). Then the physical page number, concatenated with the original low-order nine bits are the physical address sent to memory. When the map is not enabled, the virtual page number is unchanged and becomes the physical page number. Since it is only nine bits long, the contents of the HI-ABS-MA register are used as the extra four high-order bits:

```
                              Virtual Page #
    No Mapping:   I  HI-ABS-MA  I  MA 18-26  I  MA 27-35  I
                  -----------------------------------------
                   14         17 18        26 27        35


     Mapping:     I    Physical Page Number  I  MA 27-35  I
                  -----------------------------------------
                   14                      26 27        35
```

The map has two complete, separate tables, one for EXEC mode, and one for USER mode. It is implemented as a small fast memory which is 1k by 18 bits. The ten address bits are the nine virtual page number bits (MA 18-26) and the exec mode bit (see below). The 18 looked-up bits are: 13 physical address bits, one currently unused bit called concealed page, three bits for separately enabling read, write, and execute accesses, and a valid bit. The valid bit with each entry tells whether or not that entry has been loaded. There is provision in the hardware for clearing all the valid bits, so that the map can be loaded one entry at a time, as each entry is needed (demand loading). The three enabling bits (which are actually disabling bits) are compared with the type of cycle being requested to decide whether or not the access is permitted.

There is one modification to this general scheme which should be mentioned here. There is a class of operations (represented in some implementations by execute-mapped) which recuire accessing memory as though in USER mode (through the user map) even though the machine is basically in EXEC mode. To make this possible, it is not quite the case that the exec mode bit is used as one of the map-memory address bits. Insteac, there is a 4-bit shift register called the MAP-EXEC-SR, whose output is used as this address bit. The output of the MAP-EXEC-SR is, unless special action is taken, the same as the exec bit. The special action is the loading of the MAP-EXEC-SR. It is loaded, from OBUS bits 23 thru 26, by specifying it as a destination. Ones loaded into it represent USER mode, and zeros represent EXEC mode. The low order bit (bit 26) is, immediately upon loading, the user/exec bit used in referencing the map memory, and succesively higher-order bits are shifted intc that position each time the MAP-EXEC-SR shifts. The MAP-EXEC-SR shifts just before any cycle during which a memory write cycle is to be taken. This in general means that the low-order bit (26) loaded into the MAP-EXEC-SR will apply to read-type references, and the next higher bit will apply to the first write reference. Higher order bits would apply tc subsequent writes, if any. The MAP-EXEC-SR is reset to the state indicated by the real exec mode bit at the start of fetching the next instruction (macro instruction), or any time the real exec mode bit is directly changed. Or when PC Flags are changed

Map Traps :

Any time an illegal or invalid map entry is used, a map fault condition exists. This is handled by a special micro-code trap. On stores, if the relevant map entry is not present (invalid) or indicates no write permitted, then the map fault condition is true during the cycle during which the memory access cycle would have occured. If this happens, the write cycle is prevented, and a map-fault trap is taken on the next cycle. On fetches, the distinction between read and execute is determined by the MAPF field of the current micro-code instruction. If MAPF contains 0 or 1, ← don't believe this it is an execute, otherwise it is a read. If the relevant map entry is invalid, or has the appropriate prevent bit on, the read cycle is prevented. A map fault condition occurs only when the micro-code destination fielc contains FIXMAC-MAPF-RD or FIXMAC-MAPF-WRT and the entry is invalid or indicates cycle prevention. This can be the cycle during which the read cycle would have happened, or any later cycle, until the MA or the state of the map are changed. A map fault condition occurs if the destination field (of the micro-instruction) contains FIXMAC-MAPF-RD and the map entry is invalid, or indicates read (execute) prevent; or if the

DEST field contains FIXMAC-MAPF-WRT and the map entry is invalid, or indicates read (execute) prevent or write prevent. Thus, FIXMAC-MAPF-WRT is intended for fetches of locations that are going to be re-written, and FIXMAC-MAPF-RD for fetches from locations which are not.

When a map-fault condition occurs, a trap is taken by causing micro-code control to transfer to a map trap location on the next cycle. Note that this is usually two cycles after the one on which the read or write was initiated. The address of the trap location is formed as follows. The low-order two bits are 0; the next four come directly from the MAPF field in the current micro-code word; the next bit is 1; the next three are 0; and the high-order six come from the MAP-DISP register. Some examples:

MAP-DISP reg.	MAPF field	resulting trap location
0	0	100
0	1	104
0	7	134
1	0	2100
3	16	6170

Input/Output

For the purpose of attaching input/output devices to the F-3, there is a structure called the F-BUS. The F-BUS is used both for transfering data (or commands or status) one word at a time through the CPU (IOTS), and for direct memory access (DMA) in which the device transfers directly to or from the main memory.

The F-BUS consists of a bunch of wires on the back-panel which are wired in parallel to all the slots into which I/O cards can be plugged. There are 36 data lines, which are bi-directional and which are used for both IOTS (input/output transfers) and DMA. There are five device-address lines, which are used for ICTS, to tell the devices which one the CPU is coing the IOT to. There are five interrupt-address lines, which tell the CPU which device is trying to interrupt. There are 22 I/O address lines, which tell the main memory, durring CMA transfers, which memory address is being referenced. And there are a number of control lines.

IOTS are initiated by micro-code instructions which have IOB-OUT (request I/O bus output cycle) or IOB-IN (request I/O bus input cycle) in the special function field. When this occurs, the transfer takes place on the cycle following. In the case of IOB-OUT, the data in the IOD register is placed on the F-BUS data lines, and, at the end of the cycle, IOE OUT STB is sent anc the device whose address is in the DEV-ADR register will strobe the data. In the case of IOB-IN, the signal IOB DRV in will be sent during the whole cycle, which causes the addressed device to place its data on the F-BUS data lines, where the CPU sees it by selecting I/O data as external source. In addition to all this, the four bits of the MAPF field in the micro-code instruction (the one following the one with IOB-IN or IOB-OUT) are driven onto four SUB-SEL (sub select) lines of the F-BUS, and may be used by devices to distinguish intended sources or destinations within the device.

Interrupts from devices are handled as a special case of opcode dispatching. There is a common INT RQ (Interrupt Request) line, which is driven low (open collector) by any device wanting an interrupt. And there is a daisy-chained INT IT'S YOU line to establish a priority for when more than one device requests at the same time. The interrupting device sends its address on the INT ADR lines, and, after synchronization, the next dispatch will go to the interrupt location, and the INT ADR lines will be turned around and sent out on the DEV ADR lines, so IOTS will refer to the interrupting address, and A-MEM references will refer to the block of eight locations associated with that device. There is more about this in the dispatch section.

DMA cycles are initiated by the device driving the common (open collector) DMA RQ line. There is also a daisy-chained BUS IT'S YOU line which establishes a priority. The highest priority requesting device places the memory address of the word it wishes to access, and either DMA READ RQ or DMA WRT RQ on the appropriate F-BUS lines (tri-state). After the request line has been synchronized, and when the memory control decides it can do a DMA cycle, it does the appropriate type of memory cycle, to the location whose address is on the F-BUS address lines. If it is a write, it sends BUS ENABLE WRT DATA during the cycle, which causes the device to put its data on the F-BUS data lines. If it is a read, it latches the data and sends BUS READ SECOND CY STB during the next cycle, which causes the device to strobe the data from the F-BUS.

A-MEM

The A-MEM is a small memory in the CPU which is 256 words by 36 bits; it therefore has eight address bits. The A-MEM is accessed as an external data source, and its data can therefore be rotated and masked. It is stored into from the OBUS. Access to the A-MEM is controlled by combinations of things in the external source, destination, and special-function fields in the micro-code word.

There are a number of options for the source of the eight address bits for the A-MEM. In the usual case, the high-order five bits come from the F-BUS device address, which comes either from the DEV-ADR register or the INT-ADR (interrupt address) register (see I/O section). Hence, the high-order five bits are, in the usual case, the number of the I/O cevice currently being thought about. The low-order three bits, in the usual case, come from either the low-order three bits of the EXT-SRC field (fetching), or the low-order three bits of the DEST field (storing). Another option is to force the high-order five bits to be 0, with the low order three as above (SPEC [A-MEM-APR]). The final option is to have the high-order four (note! -- 4) bits come from the high-order four bits of the F-BUS device address, or be 0, as in the first two examples; and the low-order four bits come from the AC-SEL register, which is also a counter. The purpose of this is to allow transfering all of the AC's to or from a 16 word block of the A-MEM with a short and simple micrc-code loop.

Please note: the A-MEM can not be read from and written into on the same cycle!

Control Logic -- General Features

Clocks & Cycle Length

The basic (minor) clock period is 50 nano-seconds. The length of each cycle is controlled by the micro-code (occasionally overridden by some logic) and is a multiple of 50 nano-seconds. The shortest cycle which can be specified is 300 nano-seconds, and the shortest which is useful is 350 nano-seconds. Most cycles are 400 nano-seconds or longer. 400 is the shortest for cycles on which memory cycles happen. The cycle length is specified by the CY-LEN field in the micro-code. This is overridden for DMA cycles, which are forced to be the longer of 550 nano-seconds and whatever is specified by the micro-code.

Bus Control & Lock-out

Dispatches, Interrupts, I/O Stuff
(ECC & AR INT & Switches)

Plan of Attack -- Micro-coding

General Simple Instruction Plan (ADD, AND, XOR)

Jumps, MUL, DIV, IOTS

Lights & Switches

SCHEMATICS and CARDS -- WHAT'S WHERE

There are eight cards (wire-wrap boards) in an F-3 CPU, plus up to eight main-memory cards (printed circuit boards), and a number of I/O controller cards depending on the I/O configuration. The eight CPU cards are: three bit-slice boards (BS0, BS12, BS24), three control logic boards (CON1, CON2, CON3), a memory control and map board (MAP), and a micro-code memory board (MUM or MUM1).

The only difference between the MUM and MUM1 boards is the type of memory chip used to implement the micro-code memory. Either board contains 4K (by 72 bits) of micro-code memory and associated logic, and a few lamp drivers. The micro-instructions are read from one of two sources: the microcode memory (normally) or the bottom 2 rows of front panel switches (for cebugging - when the MI FROM SW switch is up). Unfortunately the micro-instructions may not be read back to the data path; however, they are displayed in the front panel lights for debugging purposes. The micro-instructions are clocked into registers on the CON1, CON2, and CON3 cards; where they are used to control the micro-machine. Micro-instructions may be written into microcode memory from the front panel switches or from the E OBUS. The microcode address lines (from the 2910 sequencer on the CON1 board) enter the MUM1 (or MUM) board and select a location (from 1 of four memory banks) which contains the micro-instruction. The microcode memory is made up of 1K of PROM and 3K of RAM (three 1K banks), all 72 bits wide.

The bit-slice boards contain the main part of the data paths -- the A-MEM, the external data selectors, part of the rotator the mask generator and masker, the PC switch recievers, the AR, the IR, the PC, the MA, the 2901's, and the selectors, drivers, and latches for the F-BUS and main memory data. The rest of the rotator is on the CON2 board. The three bit-slice boards are similar to each other, but not identical. For instance, the MA and PC, which do not have left halves, are not included on the high-order board. BS0 is the high-order board (bits 0-11), BS12 is the middle one (bits 12-23), and BS24 the low-order board (bits 24-35).

The MAP card contains (as one might guess) the map, and its associated logic. It also contains some main-memory data paths and the error-correcting-code logic, memory board timing and controlling logic, address selectors, and a few odds and ends, like lamp drivers.

The CON1 card has the clock and individual clock pulse generating logic, the timing and sequencing logic for the F-BUS, main memory, and DMA cycles, and the micro-code memory control. Also, the micro-code sequencer (PC), condition selector, and much data-path control logic are on CON1. The destination strobes are generated on CON1.

The CON2 card contains the bulk of the rotator, the

ALU carry-look-ahead logic, the MASK-SIZE register and encoder, the ROT-SIZE register and selector, the constant generator, the AC address selector, the AC-SEL counter, the ALU shift end-condition logic, the CRYOV register, the DEV-ADR register and the INT-ADR register and selection logic, the interrupt synchronizer, the A-MEM address generator, and some lamp drivers.

The CON3 card contains the data light register and drivers, some switch IOT and interrupt logic, the map enable bit, ECC error interrupt logic, and generates the ALU control signals.

Logic -- Drawing by Drawing

The logic of the F-3 is discussec in detail on a drawing by drawing basis, with the exception of a few pieces of logic which are distributec across several drawings (and, in some cases, several boards) which are discussed in sections of their own.

THE BIT-SLICE BOARDS:

Since much of the logic is the same on the three bit-slice boards, they will be covered together. The drawings for BS0 are named CFBS1 thru CFBS7. Those for card BS12 are named CFBS11 thru CFBS17 plus CF12HM, and those for card BS24 are named CFBS21 thru CFBS27 plus CF24HM.

CFBS1, CFBS11, CFBS21 -- External Data Selectors, A-MEM, and Strobe Buffers.

These drawings are the same, except for bit numbers. The 12 74S251's are the external data source selectors. The selector address bits are ESS4, ESS2, & ESS1. The A, B, and C versions are for fan-out. The 74S251's select AR, memory data, mask generator output, constant (many bits of which are GND) PC (CRYOV in left half), MA (left half zero), IOD (F-BUS cata), and IR. The A-MEM (the twelve 29721's on these drawings) are selected by disabling the 74S251's and enabling the 29721's (tri-state). The signal ENBL SRC RIGHT L (also LEFT) is the 74S251 enable; the LEFT and RIGHT versions are logically identical. The A-MEM enable is A-MEM CE L. A-MEM WE L, is the write strobe for the A-MEM. The A-MEM address bits (A-MEM ADR #) and the CE and WE signals are buffered for fan-out by the 74365's at the left of the page. Since the 29721 inverts its data, the OBUS is inverted by the 74LS04's before going into the data inputs.

At the left of this page is a tristate driver which buffers the following strobes: STB AR, STB HOLD, STB IOE, STB IR, STB MA, STB PC, and CPU RD. Some of these are clock inputs to registers, and the cthers are the gate inputs to latches.

CFBS2, CFBS12, CFBS22 -- Masker & PC Switch Selectors.

The logic at the bottom of the drawings (except for the CFBS2 drawing, where this logic is in the middle of the page), which is made of 74S08's and 74S32's is part of the mask generator and is discussed seperately, under that heading (see CON2 schematic: CFGPR description). The 12 sections of 7408, in the middle (top on CFBS2) of the drawings are used to AND the output of the rotator (SHC #) with the mask. The 74157 IC's which they feed then select between this masked version of the data, and an unmasked version. The unmasked version is selected in three situations. First, it is done when the mask is itself selected as the external data source, so that the generated word can be rotated without losing high-order bits. Secondly, it is done when the constant is selected as the external source, because the MASK SIZE field is used as the constant. In these cases, since one has control of the data (mask or constant) no masking of it would ever be needed anyway. Thirdly, the unmasked data is selected whenever a shift (within the 2901 ALU, not the 25S10 rotators) type of ALU destination is selected. This is because, to save micro-code word length, the MASK-SIZE field is used to specify the shift end conditions (i.e. rotate, logical, arithmetic, or special MULT/DIV) with these types of operations. Since ALU shift destinations are only used for multiply, divide, and double-word shifts, it was not thought worthwhile having micro-code bits dedicated to them. The outputs of the 74157 IC go directly to the ALU (2901) data inputs.

The 74365's at the top of the drawings, except CFBS2, are the PC switch selectors (drivers). When enabled, they drive the PC switch data, which comes directly from the switches, onto the micro-code address lines (MIAD #) and the F-BUS data lines (IOD #). Both of these are tri-state. These crivers are missing from CFBS2, and only partly present on CFBS12, because the relevant bits are missing from the micro-code address and the PC switches.

CFBS3, CFBS13, CFBS23 -- AR & IR.

The 2918's in the upper left are the AR. They are loaded (STB AR) from the OBUS. The 2918 tri-state outputs are used to drive the AR data (bits 20-35) onto the micro-code address lines (MIAD #) during micro-code memory stores. ENBL AR MIAD L is the enable for that. The IR is made of several different sorts of type-D flip-flop. This is so the strobes for different parts of the IR can be seperated. The strobes (EARLY LD IR STB and STB IR) are buffered through a 74S241 (on CFBS0, CFBS11, and CFBS21) before they reach the IR register clock inputs. EARLY LD IR IDX STB is not buffered, but instead comes directly from the connector to the clock input of its IR register on CFBS13. Note that some of the IR bits have two names, as, for example, IR 12 = IR AC 1. On CFBS3 are two 7485 comparators. They compare bits AR 0 thru AR 5 with bits AR 6 thru AR 11, and produce a signal which is high if the former is less than the latter. Also on CFBS3, in the lower right, is a 7425 which generates a signal which is high when bits IR 9 thru IR 12 (AC field) are zero. Likewise, on CFBS13, is a 74S20 which generates a signal which is high if bits IR 14 thru IR 17 (IX field) are not zero.

CFBS4, CFBS14, CFBS24 -- MA & PC, & Part of Rotator.

At the left of the drawing (on CFBS4, the only thing on the drawing) are three 25S10's. This is the first stage (of three) of the rotator. It takes the data from the external data selectors (E.SRC #) and rotates it 0, 1, 2, or 3 bits left, based on ROT SIZE 1 and RCT SIZE 2. The other two stages of the rotator are on the CON2 card (see description of CFSH2 and CFSH3 drawings) and complete the rotation.

Since the PC and MA have no left half, there is nothing else on CFBS4, so the following refers to CFBS14 & CFBS24 only. At the right is the MA, made of 74S174 (for speed). Its input comes from a two-way selector (74157) which selects either the OBUS, or the PC. The PC is the next thing to the left of the 74157's, and is made of 74161 four-bit binary counter chips. Its input comes from the CBUS. note that the carry chain, connecting successive 74161's, extends across the card boundary between CFBS14 & CFBS24. when PC enbl incr h is high, the pc will count (up, by one) each time stb PC makes a transition from low to high. if PC enbl incr h is low, the PC will be loaded from the OBUS on each such transition.

CFBS5, CFBS15, CFBS25 -- 2901 & E OBUS Driver

these drawings have the 2901's, with all their inputs and outputs labeled. note that out=0 is an open-collector signal -- the pullup is on CON1. things like ALU ov and out<0 are outputs of the high-order 2901 (CFBS5). at the bottom of the drawing are two 74365's which take OBUS and drive it as e OBUS onto the back-panel. this is for fan-out.

CFBS6, CFBS16, CFBS26 -- F-BUS & Main Memory Data Selectors & Latches

at the left of the drawing are two 74S373. these are latches with tri-state outputs. they latch the data coming in from main memory at then end of each read cycle (cpu rc h) and hold it until the next read. their output (mem x) is the mem input to the external data selector. the 2918's with which they are interspersed are the HOLD register. they are loaded from the OBUS (stb hold) and their output goes to the 74157 (to their right) which selects data to be sent to mem on stores. the tri-state version of the hold register is wire-ored with the read-data latches to make the mem x inputs to the external data selectors. this makes the store data available instead of the old read data.

the other input to the 74157's is the F-BUS data (IOD x), which is selected on DMA cycles (for DMA stores) -- at all other times the HOLD register is selected, for CPU writes.

at the right of the drawing are two 74S373's and two 74S374's. the 74S373's latch the memory read data on DMA reads, and drive it onto the F-BUS data lines (IOD x) when appropriate (mem to IOD l). the 74S374's are the IOD register for holding the data to be sent to the device on iots. it is loaded from the OBUS (stb IOD) and, when the iot is done, enabled onto the F-BUS data lines (cpu to IOD l). at the bottom right of CFBS16 are some gates which detect that mem 13-17 are zero, and send mem idx-ind when they are not. this is to detect the presence or absence of indexing and indirecting on words comming in from memory.

CON1 CARD:

CFCLC -- Clock Generation, Cycle Length

in the lower left are three 74S161 counters. these decide the length of each cycle, and generate a once-per-cycle signal, which later becomes the main clock pulse. each 74S161 has its carry out output inverted and connected to its carry in input. this causes each counter to count on each 20 mhz clock pulse, until it reaches octal 17, or unless its load input is low. the general scheme is that each counter is loaded with a number (by having its load input low during a clock pulse), then, when load goes high, it counts (at 20 mhz) until it reaches octal 17, at which point it quits counting, and remains at 17 until the next load. the right-most of the three 74S161's counts the first (constant) part of each cycle. its load input is low at the start of each cycle, which causes it to load an octal 12. this makes its carry out output low, which forces the output of the 74S10 (which goes to its load input) to be high, which means it will count until it gets up to octal 17 (this is six clock pulses, or 300 nanoseconds, counting the clock which loads the 12). the carry out of this counter goes to the load inputs of the other two counters. this means that during the clock pulse which counts this one to 17, the other two will load (also during earlier pulses, but they don't matter). the left-most counter loads either 17 (if no DMA cycle) or some smaller number (minimum for DMA cycles, currently 12 -- for 550 nanosec.). the middle counter loads the number from the cycle-length field of the current micro-code word. at this point the right-most counter has 17. if both of the other two counters have loaded 17, the output ouf the 74S10, which ands the three carry outs, will go low, which will start the next cycle (note that six clocks have happened -- 300 nanosec.) by causing the right-most counter to load its 12. if either of the two left-hand counters has loaded something other than 17, its carry-out will be low, and on successive clocks it will count, until both have reached 17, at which time the next cycle will start. this arrangement means that the a cycle is a minimum of 300 nanoseconds, but is lengthened to the longer of the two lengths specified by the inputs to the two counters -- DMA and cy-len.

the output of the 74S10, which is low at the start of each cycle, is strobed into the 74S74 above it on each clock pulse. this produces a synchronized once-per-cycle signal called pre-cpu clc. to the right of the 74S74 (and below) are two 74164's which form a 16-bit shift register. the high once-per-cycle signal goes into the shift input. the first output of the shift register will then be high for one clock period (50 nanosec.), and this defines CPU clock time. this output is inverted (through the 74366) to produce the actual CPU clc signal. most of the clock signals go low for 50 nanoseconds, and have their effect on their rising (trailing) edge. consequently, cpu clock time is the trailing edge of this pulse. each successive output will be high for one clock period, 50 nanoseconds later than the one before it. the shift register is cleared when the 74S10 goes low -- near the end of the cycle -- to avoid having late clocks slop over into the next cycle.

CPU clc is the first clock of each cycle, and is the main clock for timing everything in the cpu. the destination registers (with a few exceptions) are strobed by it, as is the micro-code instruction register (mi). in other words, the sequencing of the cpu is such that at the start of each cycle, the micro-code word is strobed into the mi, and then during the cycle all the data paths set up and the output of the ALU settles to the result; also the micro-code sequencer forms the address of the next micro instruction to be executed, and that word is fetched from micro-mem. when all of this has settled, the next clock pulse happens, which strobes the data result into all indicated destination registers, and loads the next micro instruction into the mi.

one-hundred nanoseconds after CPU clc on every cycle is early stb clc. this generates, when appropriate, the strobe for the left half of the ir. this is cone so that on the first cycle on which data is available on a memory fetch, it can be put into the IR and the op-code can be dispatched on. without this feature, the data would not reach the ir until the end of the cycle, and the dispatch could not happen urtil the next cycle.

also 100 nanoseconds after CPU clc is bus clc. this is the main timing pulse for the F-BUS and for main-memory cycles. since a memory cycle can't start until the address is available, which takes almost 100 nanoseconds (from loading MA to output of map), it was simplest to start all memory cycles 100 nanoseconds after CPU clc; and since the memory can't do anything for this cycle during the first 100 nanoseconds, it might as well be finishing the previous cycle. for these reasons, the memory cycle is skewed by 100 nanoseconds with respect to the cpu cycle. the F-BUS cycle is made to agree with the memory cycle to make DMA cycles simple.

at the lower right are two 74109's, whose clock inputs are driven by two 7402's. these generate, timing signals for sequencing micro-memory stores. mi sto mucode will be true if the current micro instruction indicates micro-code storing. if this is true, the top 74109 mu sto go will come on at time 300 (300 nanoseconds after CPU clc) and will stay on until time 650. the lower one, which generates the actual write strobes, will come on at time 400, and last until time 600 -- generating a 200 nanosecond write strobe. mu sto go causes address and data gating to set up for the store. below the 74109 are four gates which generate the write strobes or the appropriate half of the micro-mem, and or in the write strobe generated by the deposit switch.

just above the 74366 which buffers the clock signals is a 7474 whose output is A-MEM we clc h. this flip-flop is turned on through its direct set input by the leading edge of the time 200 clock signal -- in otherwords at time 150. the flip-flop stays on until the enc of the cycle (turned off by CPU clc). this signal (high) is used to generate the write-enable for the A-MEM chips.

the upper left and middle of the drawing has some logic fcr generating main memory refresh timing. since the main memory is made of dynamic rams, refreshes are necessary. the 74S161's count down the 20 mhz clock and generate a rising edge every 12.8 microseconds. this period means that the refresh counter (on the mem card) will count through its 128 column addresses once every 1.6 milliseconds -- which is well within the 2 millisecond limit. each time this edge occurs, it sets the first 7474 (or the second, if the first is already set). the output of this flip-flop is synchronized to create refr rq, which causes a refresh cycle to happen at the next opportunity. the purpose of the second 7474 is to detect when refresh cycles have been lockec out by DMA cycles for a full 12.8 microseconds. in this case the second flip-flop hard refr rq disables (on another drawing) DMA cycles. the signal refresh is true during refresh cycles and, if the second flip-flop rq is not on, clears the first flip-flop (through the 74S00). the first flip-flop being clear clears the others. if the second flip-flop is on when refresh happens, its trailing edge sets the flip-flop below the second one, and the seconc refresh cycle will clear the first flip-flop.

various cther clock pulses are generated by the cutputs of the shift register which are covered where they are used.

CFMCON -- Clocks, Micro-Memory Control, Control Switches

in the upper middle of the drawing is a 7474 whose lower output is labeled -cpu stop. this flip-flop when set, stops the cpu, and when cleared, allows it to run. the flip-flop is clocked at the end of all cycles during which the CPU is running. if the stop switch is on, and the micro-instruction is not requesting an F-BUS iot cycle, the flip-flop will be set, stopping the machine. iots are allowed to take both of their cycles before stopping. below the CPU stop flip-flop are two flip-flops which detect the leading edge of the continue switch, or either of the exequte switches. when this happens, the cpu stop flip-flop is turned off once per edge. note that if, at this time, the stop switch is off, the machine will run until stopped; whereas is the stop switch is on, the clock at the end of the cycle will turn the stop flip-flop on -- hence causing a single cycle to happen (or two cycles if an iot). the CPU stop flip-flop is also set (through its direct set input) by certain error conditions (micro-code parity errors, and main-mem hard ecc errors, if enabled).

to the right of the CPU stop flip-flop are the gates which and the not-stopped conditions with cpu clc to generate the clock signals which go out to the rest of the machine. there are several versions for fan-out and to make both flavors (high & low). in addition, PC clc, which controls the micro-code sequencer (next address generator) is disabled, while the others are enabled, if the xct dat switch is pushed. this is to allow executing a micro instruction (probably from the switches), and having its data effects occur, but not changing the micro pc.

at the left middle of the page is the 20 mhz clock, which is simply a ttl clock dip (square-wave generator) and some gates for drive.

the bottom half of the page has some logic for controlling the micro-code memory. since up to four mum cards (with 4k each) could be plugged in, the 74155 at the right decodes the appropriate bits of the micro-address and generates the enables for the cards. when any of the xct switches are pushed, or the sw mi from sw switch is on, the enables are all turned off, which disables the memory chips. also enbl mucode 257 l is made true, and sel 257 OBUS is made false. this causes the output of the micro memory to be driven by the data switches. at the left, the deposit switch (depo sw) is anded with the stop switch (to avoid clobbering micro-mem), and the leading edge sets the 7474. this causes its upper output to go low, and this output goes to the input of the 74164 shift register. the shift register is always shifting (at 5 mhz), so a string of zeros will be shifted through the register, until it reaches the fourth output, which turns the 7474 back on. hence a string of four zero bits will move through the shift register. when the first output goes low, it turns off the mem-chip output enables (mu oe l) and enables the data switches onto the micro-mem output bus (which is also the input bus) through the 7427 & 7402. it also forces the mem-chip chip enables on, through the 74155. 200 nanoseconds later, the second output goes low, sending the write-enable. 600 nanoseconds later, the first output will go high again, but since the third and fourth outputs are by this time low, that will have no effect. 200 nanoseconds later, the second output will gc high, turning off the wrt-enbl. next the third goes high, releasing the switch and chip enables; and finally the fourth goes high, turning the mem-chip output enables back on. this completes the ceposit switch sequencing. note that mu sto go sneaks into the 7427 and the 7410's to enable the OBUS onto the micro-mem bus, and disable the mem-chip cutputs.

CFMIP -- Micro-Code Parity, Random Signals

at the left of the page are four 93s48's, which generate parity on the bits of the micro-instruction register which are on the CON1 card. parity from bits on other cards comes in (mi par 1,2,3) and the parity of the whole 72-bit word is generated by the right-most 93s48. this if wrong (odd), is anded with the mi par stop switch, and generates mi par err h, which (elsewhere) sets the CPU stop flip-flop. at the upper middle are four 74S32's which buffer 10 bit of the external source field and send it tc the bit-slice cards to enable (if low) the external data selectors. just to the right of this, the same bit enables the A-MEM chips if it is high -- through the 74S51. if the current micro instruction specifies storing in the A-MEM (dest A-MEM), the 74S51 enables the mem-chips, and the 7400 sends write strobe, during the A-MEM we clc period.

in the middle of the page, a 74365 buffers the MAPF field, and sends it to the F-BUS as the sub sel bits. it also buffers the in enable and out strobe for iots. in the lower right are two 7474's. the upper one is cleared when stb hold happens, enabling the HOLD register onto the mem data lines, and disabling the mem data. stb ma clears the lcwer 7474, which sets the upper one, (which sets the lower one), reversing the situation (enabling mem data, instead of hold).

CFDEST -- Destination Field and Special Function Field Decode

the 74155 at the upper right decodes DEST field codes 0 to 3, and produces the early (time 100) strobes to the indicated parts of the IR left half. below it are two 74S138's, which decode DEST field codes 0 to 17 and send the strobes at end of cycle (stb clc) to the indicated data path registers. a few of these strobes are not for registers, but cause control functions to happen (like stb hi abs MA and clear mi error l). at the left are two 74S138's which decode codes 20 to 37. the bottom one decodes 20 to 27, and sends end-of-cycle strobes. the top one decodes 20 to 37 in pairs, and its output is not conditioned by the strobe pulse, so, for example, mi strt wrt l is true for the whole cycle, if either dest[20] or dest[21] (strt-wrt or mem-sto) are selected. above this are some gates which do a little further decoding. gonna memsto detects when the micro-code is doing a store, so the local user shift register (map card) can shift. the fetch next instr signal detects when the micro-code has finished with a macro instruction and is going to the next, so the local user shift register can be reset. it detects when the MA is being loaded from the PC, or both the MA and PC are being loaded (as on a jump). at the middle left, dest codes 30 through 33 generate the micro-instruction fixmac l level. this, at the bottom left, causes the HOLD register to be loaded if (and only if) the MA contains an AC address. this, if it occurs, will cause mem data to come from the HOLD register, instead of memory. the purpose of the fixmac function is to allow memory locations 0 to 17 to be the AC's. if, during the cycle that the main memory is doing its fetch, the fixmac destination is given, and the ALU is caused to fetch the AC addressed by the MA, that AC data will be strobed into the hold register, and look like it came from memory, only if the MA contains an AC address.

at the upper middle are two gates which generate the strobes for the AR and ma. in the middle are two 74S138's which decode the special function field, and send out the signals for them. note that many of these signals are caused by several special function codes, so many of the codes cause a combination of things to occur. for example, code 12 causes sel A-MEM apr, and 14 causes dest A-MEM, but 13 causes both. codes 6 and 7 cause the PC to count, if the dispatch (being done at the same time) is going to be based on the opcode (no interrupt, index, or indirect) whereas code 10 causes the PC to increment regardless.

in the upper left is the 2910 12-bit sequencer chip. it contains a 12-bit counter, and can perform a number of functions, specified by the four inputs i0 through i3. it can simply count its counter, or it can perform unconditional, or conditional, jumps. the condition input is cc, and comes from the signal cond (generated elsewhere, by a selector) which is disabled (forced false) for jump-codes (j code) 14 to 17. this is because 16 (continue) and 17 (counter load) do not wish to jump. 14 is unused and 15 (special loop) ignores the condition code. the 2910 has a loop counter, which is loaded on jump code 17, by making the -rld input low (the 74S20). it also has a stack, and can perform conditional pushj and popj. the clock for the 2910 is PC clc. the 2910 output is driven onto the micrc-code address lines by the 74S241's. the jump address for jumps and pushjs comes from the 2910 data inputs, called j adr x. the rest of the page is devoted to generating those bits. the logic om this page is most easily described in terms of what happens for each of the j-codes.

j-code 0 is the 2910 reset function, the j-adr is ignored.

j-codes 4 (unused), 6 (popj), 10 (same as 0), 11 (unused), 14 (unused), 16 (continue), and 17 (loop-counter load) ignore the j-adr. the three low-order j-code bits go directly (except for map-fault traps, see below) into three of the function select inputs, the fourth is grounded. the condition code is disabled on 16 & 17 so they won't jump.

j-codes 1 (pushj), 3 (jump), 5 (loop), and 7 (jpop) all generate the j-adr the same way -- from the J-ADR field of the micro instruction. since, for these codes, none of the used outputs of the 74S138 in the lower left are low, the two address bits into the 74153's which select j-adr bits 2 through 9 will be 0, causing them to select micro-instruction j-adr. the two low-order address bits into the 74151's (for bits 10 & 11) will be low for the same reason, causing bit 10 to select micro-instruction j adr 10, and bit 11, since mi j code 1 is high, to select micro-instruction j adr 11.

j-code 2 (lbjump) is identical to 3 (above), except that since micro-instruction j code 1 is low, j-adr bit 11 (the lower 74151) will select cond, causing the jump (which is unconditional) to be to the indicated address, or that address plus 1, depending on the condition.

|L

j-code 12 (dispatch) is the most complicated case. the 74153 address will be 2, because the 74S138 disp output will be low. this will cause j-adr bits 2 to 9 to come from the IR opcode field, bits 0 to 7, except that for certain opcodes, IR bits 6 and 7 are forced to 0, and except in the case where the ir contains the indirect bit, or a non-zero index field, or an interrupt is pending. if any of these is true, the enables of the 74153's are forced high, causing their outputs to be zero, by the 74S11 and 74s32's (lower left). j-adr bit 10 (74151) comes from IR bit 8 (the low-order opcode bit) when none of the special conditions exist, (except that ir8 is forced to 0 for some opcodes). if any of the special conditions exist, the bit comes from the (inverted) state of a signal which amounts to the console stop switch wants an interrupt. the low order j-adr bit (11), comes from the interrupt signal, in all cases. note that this means it will be zero, except in the special case of interrupts; thus IR dispatches have the low-order bit 0 (only go to even addresses). note also that in the case where there is indexing or an interrupt, j-adr bit 9 is forced to 1. the effect of all this special stuff is to generate the dispatch addresses (as covered in the micro-code manual) 1 through 7, for the special cases.

j-code 13 (special dispatch) is very simple. it jumps, conditionally, to the low-order bits of the CBUS. this allows jumping to generated addresses. the 74153 adoress bits will be 01, which will cause them to select the OBUS; likewise the 74151's.

j-code 15 (special loop) is likewise fairly simple. the 74153 address is 0, as for jumps, and these bits select the micro instruction J-ADR field, as for jumps. the 74151 address will be 7 for both low order bits, causing bit 10 to come from cond, and bit 11 to come from the J-ADR field. this means that special loop, if it jumps (cepends on the loop counter), jumps to j-adr or j-adr plus 2.

the final case is that of map-fault traps. when map oops disp is true, the 2910 function inputs are forced to unconditional jump, the 74153 address is forced to 3, selecting the micro instruction MAPF field, (or 0 or 1), and the 74151's are disabled, forcing the two low-order bits to 0. this causes micro-code control to be transfered to the address 100+mapf*4 (except for the high-order bits, bit 1 and above, covered later).

all of the above covers j-adr bits 2 to 11. the higher order bits are on the next drawing. see below.

CFHMAD -- High Order Micro-Code Address Bits

at the bottom middle is a 74153 which selects the j-adr bits, 0 and 1. in all cases except map-fault traps and special dispatches, it selects the micro instruction J-ADR field. note that this means that even dispatch gets these bits from the J-ADR field. in the case of special dispatch, it selects, as one might expect, the OBUS. in the case of map-fault traps, it selects the two low order bits of the map-fault dispatch register. the map-fault dispatch register is the 74174 to its left.

above the 74153 is a 74157, which selects the rest of the map-fault dispatch register in the case of map-fault traps, and selects the appropriate bits of the OBUS at all other times. the output of the 74157 goes to a 74173 four-bit tri-state register, which holds the high-order (negative numbered) bits of the micro-code addrss. it is loaded only on map-fault traps and special dispatches, nothing else can change which page you are in. (except reset, which clears the register). the output is enabled onto the micro-instruction address lines, except when the sw enbl PC sw switch is on, which enables the pc switches onto the micro-instruction address lines, or during micro-code stores, when enbl AR miad L causes the ar to be enabled onto those lines. note that this is sequenced by mu sto go from cfclc.

in the upper left are some gates which detect certain groups of opcodes in the IR, and cause ir bits 6, 7 and 8 to be ignored on dispatches. the 000 (uuo) and 300 (jumps & skips) groups will ignore all three bits; the 600 (t-type) group will ignore 6 & 7.

CFCOND -- Condition Selector and a few random things

in the middle of the page are two 74S151's which select, from the micro instruction COND field, one of 16 conditions. zero selects hi (true), 1 selects interrupt, etc. most of the conditions are self explanatory. condition 2, ma-ac, is true if ma bits 18 to 31 are zero, meaning the MA contains an ac address. condition 3, ac=0, refers to the AC field of the IR being 0. condition 12, referred to as jcond in the micro-code, uses bits from the IR opcode field, as well as the out=0 and out<0 and ALU ov signals to generate the condition for a whatever flavor of jump, skip, or t-type instruction is being executed. the condition is true if the instruction should jump or skip, and false if it should not. the output of the selector is xored with the reverse cond bit of the micro instructon.

at the right is a component carrier which has the pull-up resistor for the out=0 lne (which is open-collector) and another pull-up which generates hi for the card. it also has a capacitor to de-bounce the continue switch.

at the lower left are some gates which generate the main-memory read and write requests, and the cycle-type signals for the map to compare with its enables. read request is generated whenever the ma is strobed, except if strt wrt is true, in which case write request is generated. if neither is true, neither request is generated. if gonna wrt is false, either xeq cy or read cy is generated (except if the map is turned off). read is generated unless the micro instruction MAPF field contains zero or one, in which case xeq is generated.

CFBUSC -- Main Memory Cycle and F-BUS Control

at the end of each cycle, the state of the four requests (mem read & write, & iob in & out) are strobed into the 74175 at the upper left. this happens at the same time that the next micro instruction is loaded into the mi. the outputs of this 74175, along with some other signals, then go through some logic, which sets up for the next 100 nanoseconds, to decide what the memory and F-BUS will be doing this cycle. at the end of the 100 nanoseconds, bus clc happens, which strobes the results of those decisions into other registers. to the right of this 74175 is a 74S175 which holds the DMA and CPU memory-cycle states. its inputs come from some 74S11's. DMA read will happen this cycle, if DMA rd rq is true (from the F-BUS), and it is not prevented by anything. DMA write will happen this cycle if DMA wrt rq is true, and it is not prevented. any CPU memory cycle request or any CPU iot request (because it uses the F-BUS) will prevent DMA cycles (through the 74S02's and 74s11 below and to the right of the 74175). also, they must be enabled by enbl DMA rqs h (see below). in addition, if the next (!) micro instruction requests any F-BUS iots, DMA reads will be prevented (the 74S02 just below the 74175). this is because DMA reads take two cycles (one to fetch the data, and one to put it out onto the F-BUS and have the device strobe it). DMA cycles are also inhibitted if hard refr rq is true, so that refresh cycles can happen.

CPU memory cycles (read or write) happen whenever they are requested, unless there is something illegal about the map. along the bottom of the page are three more 74S175's which hold various other information about the type of cycle being done. refresh cycles happen if one is requested, and the last cycle was not a refresh (they cannot happen on successive cycles) and no DMA or CPU memory cycle is going to happen. they are permitted to happen at the same time as iot cycles, because they don't use the bus. there are several versions of the refresh signal. the ones labeled simply refresh is for local, on-card, use. the ones labeled refresh cy are seperate partly for fan-out. they go to the memory cards. short refresh cy goes to the map card and is seperate for noise and far-out reasons. both it and refresh cy h are cleared by the leading edge of CPU clc (150 nano-seconds before the next bus clock) to avoid their slopping over into the next cycle. CON1 latch clc is set (from hi) at bus clc, and cleared 150 nano-seconds before the next bus clc, to hold a latch during that period. F-BUS iots (IOB-IN & out) happen whenever requested. DMA rd second cy is true during the cycle following a DMA read cycle, during which the read data is being put on the F-BUS and should be strobed.

|L

at the top middle is a latch, made out of a 74s64 and twc inverters. the name enbl MA IOA, is no longer as meaningful as it was before the data path logic gct changed. the latch is open (feeds its input to its output) from just before CPU clc time, until just after bus clc time, at which time it is latched into its current state and held. it decides whose address (cpu or F-BUS -- for DMA) should be used by the memory for this cycle. if either type of DMA cycle is going to happen this cycle, the output of the 74S64 is forced low, enabling the F-BUS address. otherwise it will be high, enabling the MA (and map). this is done as a latch because the signal must set up before bus clc, and must last well into the cycle, after some of its inputs may have changed. at bus clc, the latch latches until near the end of the cycle.

near the middle of the page is another 74S64. its output is low if a CPU or refresh cycle is going to happen this time. the 74S74 which it drives is strobed at bus clc, and if the output of the 74S64 is low at that time, the flip-flop output will go low, causing edge strt pulse to go high. this creates the rising edge sent to the map card, and thence to the memory cards, which causes their cycle to start. the flip-flop is clear 100 nanoseconds later by mem strt clr pulse l. below the 7400 which generates edge strt pulse is a 7474 which provides the same starting-edge production function for DMA cycles, only 100 nanoseconds later.

in the upper right corner is some logic which provides sequencing for DMA requests. when nothing DMA-ish is happening and there are no DMA requests, BUS RQ STB is sent every cycle at CPU clc time. This clock signal is used by the device to stobe a flip-flop with the state of his internally generated DMA request. bus rq stb therefore provides synchronization of requests. if on any cycle, any device sets its request, it will immediately pull down some rq l (open-collector) which then prevents further bus rq stb pulses from being sent. the 7474 generating enbl DMA rqs is set each bus clc, but cleared by each bus rq stb, so that as long as bus rq stb is being sent, DMA rqs will be ignored. now the rq situation is stable, with some device or devices having their rq flip-flops set, and the priority caisy chain can decide who will get the bus -- most of a cycle is allowed for this. when the device has decided it gets the bus, it must send either DMA read rq or DMA wrt rq, along with the memory address. the next bus clc, slightly more than one cycle after the last bus rq stb to be sent, will be the first time that DMA rqs have been enabled. at that time, or some subsequent cycle, the cycle-type logic decides the DMA cycle can happen. during the cycle, bus DMA cy is true, and the device sees it and clears its internal request level (because the cycle is being done). at the end of the cycle, bus rq stb is again sent, which strobes and clears the request flip-flop, and we are back at go.

CFMI -- Micro Instruction Register

this drawing contains all bits of the micro-code instruction register which are on this (CON1) card. the register is made of 74S174's, and is strobed by micro-instruction clc. certain bits of the register are cleared by reset, to insure that the micro-instruction contains a nop with respect to data (dest and spec-fun =0) and a reset for the 2910 sequencer, to clear its stack. other bits of the micro-instruction are on the CON2 and CON3 cards.

CFLT1 -- Lamp Drivers

what can one say about lamp drivers.

MAP CARD:

CFMAP4 -- Main Memory Control

in the upper left, edge strt pulse comes onto the card and goes into a delay-line. this is to allow address bits, etc. a little extra time to set up. the 20 nanosecond tap of the delay line is used, and is called mem strt pulse. below the delay-line, it goes into some 74S11's which and it with the true and false flavors of address bits 17 and 18, and send a start pulse to the addressed pair of memory cards (each pair of memory carcs holds 128k words). note that short refresh cy -- meaning this is a refresh cycle (short refers to the length of the signal, not the length of the cycle) -- causes the start pulse to be sent to all the memory cards, so that all the mem chips will be refreshed.

to the right of the first delay-line is another delay-line. mem strt pulse goes into this one and is delayed a variable amount to produce a pulse whose timing is right for write-strobe. this pulse is anded with mem wrt type h, which is true for write cycles, in the 74S00. the output of this is inverted and called mem wr enb h which is the write-strobe sent to the memory cards. below the 74S00 is a 74S02 whose output will be low for refresh and write cycles, and high for read cycles. this signal and the delayed mem strt pulse, set the rd go flip-flop (7474), which is used to decide whether or not to check parity (ecc). below all this are some 74s11's which and the flavors of the high-order address bits, and not-a-write-cycle to produce one signal per pair of memory cards, which is high if that pair is doing a read (or refresh) this cycle. the 74S11 outputs are strobed into a 74S175 register by the delayed mem strt pulse and the outputs of this register are the tri-state output enables which enable the read-data output of the appropriate pair of memory cards to be driven onto the memory data bus. this register is cleared, and held clear, on write and refresh cycles (on write cycles, the bus has the write data).

CFMPAD -- Address Selectors

at the bottom of the page are three 74S241's which, on DMA cycles (enbl MA IOA false), drive the F-BUS address (IOA) onto the common memory-card address lines (mem p adr x). the high-order address bits, bits 15-18, have both true and false flavors, because they are decoded, on the map card, to decide which pair of memory cards is being addressed. in the upper left is a 74S258 which inverts and drives these bits from the IOA onto the low mem p adr lines. below it are two 74365's which drive MA bits 27-35 onto the mem p adr lines on non-DMA cycles. at the right edge of the page are three 74S257's which select between the MA bits 19 to 26 and the map output bits 19 to 26, based on whether or not mapping is turned on. their output is driven onto the mem p adr lines on non-DMA cycles. to their left are a 74S257 and 74s258 which perform the same function for both flavors of bits 15 to 18. note that, since the MA is an 18 bit register, bits 15, 16 an 17 come from the 74175 (upper middle), which is the hi abs MA register. it is loaded from the corresponding bits of the OBUS when it is indicated as a destination.

CFMWDT -- Write Data Drivers

in the top half of the drawing are six 74S241's which drive the write data, including the seven generated ecc bits (see below) onto the common memory data bus (mem out), on write cycles. below these are some gates which generate the enables en w dat for them. in the lower right are some lamp drivers.

ECC -- Error Correcting Code -- a few words:

the purpose of the error correcting code used in the F-3 is to correct all single-bit errors, and to detect all two-bit errors (as well as some others). this is done by having extra bits to. since there are 36 data bits, which is slightly larger that 2**5, but less than 2**6, the single-bit correcting mechanism will need six extra bits -- since it will have to be able to address one of 36 bits. to detect two-bit errors will require another extra bit, making seven altogether. this makes a total (data plus extra) of 43 bits.

now any set of 2**n bits can be divided evenly into two sub-sets in n different ways such that no two bits are elements of exactly the same set of sub-sets. that is to say, if a set of 2**n bits is divided evenly into two subsets, say a and b (each having 2**(n-1) bits), then if the the orriginal set is divided into two new subsets, say c and d, such that half of the a's are in c and half in d, and half of the b are in c and half in d; and if this process is continued until n sub-set divisions have occured, then each bit will have a unique set of letters associated with it. there are several ways of doing this division (all of them equivalent). the simplest is to number the bits, in any arbitrary order, from 0 to 2**n, then represent the number of each bit in binary. next, there is a sub-set division for each bit of the bit-number, a given bit going into one subset if the appopriate bit of its bit-number is a one, and the other sub-set if it is a zero. since each bit has a unique bit-number, each will be in a unique set of sub-sets.

now if the orriginal parity of one of each pair of sub-sets was known, and a single bit changes, the parity of only those subsets which the bit is in will change, and, since the sub-set division is unique for each bit, from the combination of which sub-sets are in error, it is possible to tell which bit changed.

to be more specific, in the F-3 each bit is assigned (temporarily) a 6-bit number. this number is unrelated to the regular left-to-right bit numbering. the six ecc bits are included, and are given the numbers 01, 02, 04, 10, 20, and 40 (in octal). since there are fewer than 64 bits, some numbers will be assigned to no bit. these non-bits are assumed to be always zero. now the bits are grouped into six groups (not groups in the mathematical sense), based on whether or not the appropriate bit of its bit-number is on. there is a 01 group which has all those bits whose bit-number is odd. there is an 02 group with those bits whose bit-number has the 2 bit on, and so on for the 04, 10, 20, and 40 groups. note that each group has one ecc bit, and each ecc bit is in exactly one group. now, when each word is stored, its ecc bits are generated in such a way that the parity of the group is odd. the bit-number 00 is not assigned to any existing bit. when the word is read back, if any one bit has changed, one or more of the groups will have incorrect parity (since there is no bit 00). if only one group has incorrect parity, it was an ecc bit which changed, as they are the ones whose bit numbers have only one bit on; all other bits are in at least two groups. if the six error bits are given the weight of the group (01, 02, 04, etc.) and summed, the result will be the bit-number of the bit in error.

two-bit errors are detected by having a parity bit for the whole 42-bit word (data plus ecc bits). if two bits are in error, their error bits cannot cancel each other, since each generates a unique pattern. to put it another way, there must be at least one ecc group to which one of the bits belongs, and the other does not. therefore, there will be at least one ecc error bit generated. on the other hand, since two bits have changed, the whole-word parity will be correct. the and of these two conditions detects a two (or more) bit error.

CFMECC -- ECC Bit Generators

this drawing contains six parity networks; each generates the parity for one ecc group. since the main memory data bus (mem out) is common for reads and writes, these generators work for both, the only difference being that on reads, the ecc bit is included in the calculation, for checking, and on writes it is not, since it is being generated.

CFMDMX -- Bit-in-Error Logic

on this page are the decoders which generate, from the six ecc group parity checkers, the bit-in-error signal. there is one output for each bit (36 outputs) and the output (correct bit x L) is low if that bit is the one in error, and high for all the other bits. only one bit can be low.

CFMXOR -- Error-Bit Correctors

this drawing has 36 inverters (because the correct bit x l signal is low-true) and 36 xor's. each correction bit is xored with the data bit, so that the bit in error will be inverted, and all the others left alone. the inverter inverts the data bit, instead of the correction bit (which is equivalent because of the logical properties of xor) because the data bit is available first -- this way it's faster.

CFMERR -- ECC Error Signal Logic

it is the nature of the xor function that the xor of a bit with itself is zero, indepenent of the state of the bit. indeed, the xor of a bit with itself any even number of times is zero, and the xor of a bit with itself any odd number of times is equivalent to the bit itself. therefore, any xor of lots of bits, which has some bit entered more than once, is subject to simplification, by having entries of the same bit canceled in pairs. in the upper left of this drawing is some logic which generates the whole-word parity. this parity is, in principle, the xor of all 42 bits, data and ecc. but each ecc bit is the xor of some of the data bits. consequently, the whole-word parity is the xor of the 36 data bits, some of them entered more than once. this is equivalent to the xor of all the data bits which are entered an odd number of times. the logic in the upper left forms the xor of just those bits, and this is the whole-word parity bit (sto word par).

since on read, the ecc bits are just bits, not xors, regardless of how they were orriginally generated, they must be included in the parity calculation. therefore, all 36 data bits must be included also. to save logic, the data bits going to the ecc bit generators were distributed between the two parity chips (cfmecc drawing) for three of the ecc groups in such a way as to have the ones not included on the cfmerr page go to one of the parity chips, and the others to the other, so that those three chips would generate the xor of the left-out bits (except for bit 3). this is the source of the wp0, wp1, and wp2 signals. at the bottom left is the xor of the ecc bits, the whole-word parity bit, and data bit 3. the 74S280 in the middle of the page forms the xor of all these things, to generate the whole-word parity check for read.

to the right, the six ecc error bits are ored together, to generate a signal which is true if any error occurs (ecc error h). this is anded with word par odd h and the rd go flip-plop to produce the ecc multiple error h signal, which, after being synchronized, and if the switch is on, halts the machine.

CFMSAV -- Error Save Register

the register chips on this page strobe the state of the ecc error bits, the hard-error signal, and the memory address, on every cycle until they strobe one that actually has an error, which they hold onto until cleared by the micro code (clear mi error l). this makes the information about any ecc error which occurs available to the micro-code so it can log the error if it wishes.

CFMAP3 -- Data Selectors & Drivers for IOTS to the Map, etc.

the 74LS257's on this page select, in one case the error save register (see above) and in the other case the output of the map, and a few random bits. the choice is based on the sub-sel 2 bit. this data is driven onto the F-BUS data lines when an input iot is done to device number 1. this is how the micro-code looks at this stuff.

CFMAP1 -- Map Memory and Logic

this page has the memory chips for the map, except the valid (or entry present) bit. they are 2125 1k ram chips. the address comes from the high-order nine MA bits, and local user. local user comes from the 74S194 shift register at the top of the page. its purpose is described in detail in the map section above. the 74S194 is clocked every cycle, but on most cycles the control inputs are both low, and the chip does nothing. exceptions are: when gonna memsto is true (meaning the memory is doing a CPU write) the upper control input is high, and the shift register shifts one bit; when the shift register is indicated as a destination (dest map sr), or the CRYOV register is indicated as a destination (mi dest CRYOV), or the logic on CON1 decides the next macro-instruction is being fetched, both control inputs will be high, and the shift register will load. the data loaded comes from the OBUS, when the shift register itself is the destination, or from the CRYOV user bit (or the data headed for it) in the other cases.

at the right middle is a 74S64 which decides, from the type of cycle signals, and the type of cycle prevent bits from the map, whether or not the cycle is legal to the current address (legal map h). above this are some gates which generate the map oops disp signals. this is the or of a number of conditions, andded with the MA is not an ac. the conditions which cause the oops, from top to bottom, starting with the first significant 74S64 inputs, are: write cycle and map turned on and writes illegal; map trap requested (mi mapf if) and write trap requested (mi dest 1) and map on and writes illegal; map trap requested and map not valid (no entry); write cycle and map not valid; or map trap requested and cycle type illegal (legal map h).

CFMAP2 -- Map Valid (entry present) Bit, Map Clear, MA-AC

in the lower right corner of this drawing are gates which and together the low level of MA bits 18 to 31. this and forms the ma-ac signal, which means that the ma has an AC address.

across the upper right are eight 29721 256 bit rams with their outputs (tri-state) tied together. also, their data inputs and address inputs are tied together. one of the address inputs is grounded, so each uses only 128 bits. 128 times 8 is 1k. these store the valid bits for the map. when the bit is high, there is a map entry at that location. during normal operations, the 74S157's select the MA as the address (ma bits 25 & 26, and local user, are decoded to form the chip-enables) and the output will be the bit to tell if the entry is present. if an entry is stored (map wrt pulse) the bit addressed by the ma will be written as a 1 (unless OBUS bit 9 is on). indicating presence of an entry.

the exception to all this occurs when it is time to clear the whole map. the 74S161's at the left are normally held at zero by their clear inputs being low. when mi clr map l occurs, the first 74109 will, on the next 10 mhz clock edge, be cleared. this causes the map clr condition to become true, which, through the 74S02's and 74S51's below the mem chips, causes all eight mem chips to be enabled at the same time. the upper output of the 74109 going low causes the write enable to go true (low) to the eight chips, and also causes the address selectors to select the 74s161 counter instead of the ma. 100 nanoseconds later, the second 74109 will be cleared. its lower output going high forces the data inputs to the mem chips to go high (zero) and also releases the 74S161 counter, which counts at 20 mhz. this condition lasts until the counter's eighth (high-order) bit comes on, which means it has counted through all 128 addresses. when the high-order bit comes on, the first 74109 is set (on the next 10 mhz clock) which releases the mem-chip write-enable and chip-enables. 100 nanoseconds later, the second 74109 is set, which releases the address selector (back to ma) and the data input. now everything is back to normal. note that the scheme (which works) for writing zero's into all bits of each memory chip is to hold its write enable true, while cycling through all the addresses. this works, in spite of technically violating various set-up and hold time specifications, because the same data is being written in all locations. the set-up and hold specs are to ensure that when writing into a location, other locations are not changed.

in the middle of the page, an 8097 (which is the same as a 74367) buffers the no map l signal, which is the map-turned-on level. this, at the top of the page, causes the no entry level to be driven false when the map is turned off.

CON2 CARD:

CFSH2 -- Another Third of the Rotator

this drawing has eight 25S10's, and two 74S153's, which take the output of the 25S10's (on the bit-slice boards) which rotate the external data by 0, 1, 2 or 3, and they rotate it by 0, 4, 8 or 12.

CFSH3 -- Another Third of the Rotator

this drawing has eight 25S10's, and two 74S153's, which take the output of the previous page, which has been rotated by some amount from o to 15, and they rotate it by 0, 16, 32, or 48.

CFACSL -- AC SEL Register, Mask Size, Constant Generator

at the upper left is a 74161, which is the AC-SEL register/counter. it is loaded (on micro-code command) from the OBUS. it also counts (on micro-code command). below it is a 74283 4-bit adder chip, which forms ac+1 -- it takes the 4-bit contents of the IR AC field, and adds one. to the right are four 74153's which select the AC addess to be used. they are directly under control of the micro instruction AC SEL field, and select the AC-SEL register, IR index field, low order MA, IR AC field, or AC+1 adder. note that the A and B AC SEL's are the same, except for code 1, which causes a to be the IR index field, and b to be the ma. this is because the index is never used as a store address -- all AC stores in the 2901 are to the b address.

in the upper right are two 74S157's which select between the micro instruction MASK SIZE field, and the MASK SIZE register (the 74174). below them are some gates which, with the bottom half of the lower 74s157 generate the constant. when the mask sel r bit is off, the constant is just the micro-instruction mask field. when it is on, the constant is 1,,1 (bits 17 and 35 on). the other 29 constant bits are always zero, and are grounded at the bit-slice boards.

CFCRY -- CRYOV Register and ALU Shift End Conditions

at the top of the page is the logic for the 2901 shift end-conditions. the 2901 is capable of loading its destination AC with the result shifted by one, left or right. at the same time, it can shift the q register by one in the same direction. the left (high-order) end of the q register and the right (low-order) end of the AC data are connected together for shifting. therefore, for shift purposes, the q and AC are a double word, with the AC the high-order part, and the c the low-order part. the 74S253 selects the data to go to the high-order ac, or the low-order q (depending on the shift direction). the selection is based on the low order two bits of the mask field. zero selects rotate mode (high-order AC from/to low-order q), one selects arithmetic mode (zero into low end, sign bit duplicated into high end), two selects logical mode (zeros), and three selects mul/div mode. in mul/div mode, low-order q gets the sign bit (high-order ac) inverted (for divide), and high-order AC gets the xor of the sign and overflow (for multiply). the shift connection pins on the 2901 are tri-state, and are inputs if the shift is in one direction, and outputs in the other. the direction is determined by the alu d2 control bit, so this bit also determines which section of the 74S253 is enabled.

the lower half of the page has the CRYOV register. bits 4 throught 12 (except 10, there is no bit 10) are simply loaded from the same bit of the OBUS when the CRYOV register is indicated as destination (stb CRYOV). also, bit 4 (CRYOV half) can be cleared by a micro-code special function (clr half l). the high-order four bits are loaded in two different ways. if CRYOV is indicated as a destination, micro-instruction dest CRYOV l will be true (low) which will cause the 74157 to be enabled, and to select OBUS, which will go to the j inputs on the 74109's. also it will enable OBUS into the -k inputs, through the 7432's. on the next clock, the CRYOV bits will be set from the OBUS. if the load CRYOV special function is indicated, the micro-instruction ld CRYOV l signal will be low, enabling the 74157 and causing it to select the ALU overflow and carry information.

this then goes to the j inputs, while the -k inputs remain high. on the next clock, any 74109 whose input is true will be set, but none of them will be cleared (the new data will be ored into the register). if neither of these functions is indicated, the 74157 and the 7432's will be disabled, so the j inputs will all be low, and the -k inputs high, and the flip-flops will not change.

CFGPR -- ALU Carry Look-Ahead, Rotate Size, Part of Mask Gen.

Mask Generator:

the upper part of the drawing has some 74S08's and 74s32's which are the first part of the mask generator. the group of gates on the right turn the low-order three bits of the mask-size from a binary number into a unary number, as follows (numbers in octal):

binary	unary
0	0
1	1
2	3
3	7
4	17
5	37
6	77
7	177

these seven unary bits are mml 1 (low-order) through mml 7 (high-order). the gates in the left group do the same thing for the high-order three bits of the mask size (when left only is not true). so, for example, mmh 10 is true if any of the three bits is on, which means it is true if the mask size is 10 or larger. mmh 30 is true if the mask size is 30 or larger, and so on. since there are only 44 (octal) bits in a word, mmh 50 is the largest of these which is needed. these signals are sent to the bit-slice boards, where some logic generates the actual mask bits. this is done as follows. the word is divided into 8-bit groups, starting at the low order end. so the low-order eight bits are in the 00 group, the next eight are in the 10 group, the next in the 20 group, and so on. the 40 group is incomplete, containing only the high-order four bits. a given bit in a group is on if the corresponding mmh signal is true, and the corresponding mml signal is true, or, the whole group is on if the next-higher mmh signal is true. the mmh 00 signal is considered to be always true. so, for example, the 23 bit (which is the bit 23 octal from the right end, bit 35 being the 1 bit) will be on if mmh 20 is true (meaning mask-size is at least 20) and mml 3 is true (meaning the low-order three bits of mask-size is at least 3, and therefore mask-size is at least 23) or it will be on if mmh 30 is true (meaning mask-size is at least 30.

in the lower left of the page are three 74S182 look-ahead carry chain chips, hooked to the ALU generate and propagate and carry-in in the standard way. in the lower right are the 74S157 rot-size selectors, which select either the ROT-SIZE field from the micro instruction, or the rot size register, which is the 74174.

CFION -- Device Addres Register & Drivers, A-MEM Address, etc.

across the lower left of this drawing is the logic which handles I/O interrupts. when any device wishes to interrupt, it pulls down iob any int l, which is open collector. if no other interrupt is going on at the time, the next CPU clc will set the 7474 whose output is labelled interrupt. this is the signal which goes to the dispatch logic, causing the next dispatch micro instruction to go to micro-code 7, instead of somewhere based on the op-code in the ir. intr disp h (at the left) is true on the cycle when that dispatch happens, causing the 74109 to be strobed, and set (since interrupt is true). this makes its lower output low, clearing the interrupt flip-flop and causing the 7474 next to the 74109 to be cleared at the next CPU clc, making further interrupts invisible until this one has been handled. the interrupt micro-code, when it is through thinking about this interrupt, will go back to fetching instructions, and dispatching on them. this next dispatch again strobes the 74109, and clears it. one cycle later the 7474 next to it will be set, making the next interrupt visible. above all this logic are two 74157's which select the address to be sent out on the F-BUS device address lines (ion). normally this is the address in the DEV-ADR register, which is the upper 74174, which is loaded from the OBUS when indicated as a destination. when an interrupt is in progress, as indicated by the 74109 being on, the int adr, which was sent by the interrupting device, and latch into the lower 74174 when the 74109 was set, is used. note that the result of this selection is not only sent out as the ion, but also becomes the high-order A-MEM address bits (unless the special function A-MEM apr is true, which forces them zero). at the right are two 74153's which select the low order A-MEM adr bits from the micro-instruction DEST field, micro-instr SRC. field, or the AC-SEL register, based on the specification in the micro instruction.

CFC2MI -- Some More Micro-Instruction Bits

here are the micro-instruction register bits which are on this card, and two 93s48's which generate their parity. also the drivers for the ext. src. bits sent to the bit-slice boards.

CFLT11 -- Lamp Drivers

ok, lamp drivers.

CON3 CARD:

CFLT3 -- Data Lights, Register and Drivers

this drawing has the 36-bit register whose output goes to the data lights (lamp drivers attached). it is loaded from the OBUS every 100 nanoseconds if the sw look at OBUS switch is on. this allows looking at the OBUS every cycle when single-stepping the micro code, or whatever. with the switch off, the register is loaded when an iot is given by the micro-code (stb sw1). this allow the micro-code to display only selected data. for example, the function of an examine switch can be simulated.

CFSWD -- Data Switch Selectors, Reset

at the left are six 74365's which drive the data switches onto the F-BUS data (IOD) lines when the appropriate iot is given (enbl IOD i sw l). at the right is the power-reset circuit (consisting of transistors and a slowly-charging capacitor) on a component carrier. its output is low for a few seconds after power comes up, then goes high. this is ored with the clear switch, to produce reset l, which has lots of drive.

CFLT4 -- More Lamp Drivers

more lamp drives.

CFC3CN -- Back Panel Connections

these should get moved onto the other drawings, where each signal is used. their being here is an artifact of the way back-panel wire lists were generated in the dark ages.

CFIOCN -- IOT Decodinc, Some Micro-Instruction bits, and Map Enable Register

at the left are the decoders which detect the special iots (device 0 and 1) which refer to CPU functions. these include reading the data switches, reading the map output and ecc error register, strobing the data lights, etc. above is a 74365 which reads (on an iot) the interrupt address from the F-BUS.

at the lower right are some more bits of the micro instruction register, and the 93s48 which calculates their parity. above this is the 74175 which holds the no map level (turns the map on and off), as well as the bits which enable interrupts on arithmetic overflow, and ecc errors. this register is loaded by an iot, and cleared, turning off the map and disabling the interrupts, by reset. to its right are some gates which generate the s1 and s4 alu control signals. normal (when ALU d4 is low) these come directly from the corresponding bits of the mi (except that s1 is inverted -- a fact compensated for by the micro-code assembler). ALU d4 is low for most alu operations. it is high for the shifting operations, which are used only for the double-word shifts, and multiply and divide. when d4 is high, ALU s4 is forced high (the other functions are not useful when shifting) allowing the mi ALU s4 bit in the micro-instruction to be used to select whether the s1 bit comes from the s1 bit in the mi, or from the low-order bit of the q register. the effect of this is that by specifying the correct things in the ALU fields, the ALU will add either 0 or the data input to the ac, based on the low-order q bit. since a shift is also performed, this is exactly the function needed for multiply, and allows it to proceed at the rate of one bit per micro-instruction.

CFSWI -- Console Switch Control

the console switches come in here. the stop switch is debounced (left middle), ored with the ecc error interrupt and arith. overfl. interrupt, synchronized, and sent to the dispatch logic, where is causes dispatches tc go to location 4 or 5, so the micro-code can handle the problem. the other console switches (except reset), have their rising edges detected by the 7474's. the reset switch comes in directly, no edge is detected. each 7474 will be set by the rising edge of its switch, so the switchve will have to be released and pushed again to have its effect again. the flip-flops, and reset switch, are ored toghether to produce an interrupt signal.

it is synchronized, and causes an normal I/O type interrupt. at the bottom right is a 74365 which drives the interrupt address of 1 onto the int-adr lines when this interrupt happerns. the micro-code, when getting one of these interrupts, reads these switches with an iot (enbl sw IOD l) to tell which switch was pushed, then must give another iot (stb sw3) to clear the flip-flops, so they will be seen next time a switch is pushed, and not before (except reset, which will be seen every cycle).

```
(DFMATH.SLO;1 2.8) {q-odd}
TRYM6.A: ALU[MEMAC] DEST[AR] COND[Q-ODD] LBJUMP[like..mul6] $
TRYM6.M: RE-XCT D[MEM] DEST[AR] COND[Q-ODD] LBJUMP[like..mul6] $
        ALU[AC+0] DEST[D4] ENDCONN[LSH] COND[q-odd] LBJUMP[. + 2] $
        D[AR] ALU[AC+D] DEST[D4] ENDCONN[LSH] COND[q-odd] JUMP[. + 2] $
(DMOV.SLO;2 3.7) {q-odd}
        DFRQ DF-FROM-IR D[MA] ALU[D+1] DEST[Q] COND[Q-ODD] LBJUMP[DMOVEM.MEM] $
        DFRQ DF-FROM-IR D[MA] ALU[D+1] DEST[Q] COND[Q-ODD] LBJUMP[DMOVNM.MEM] $
Searching <SDD>DMOVEX.SLO;6
(DMOVEX.SLO;6 2.7) {q-odd}
        DFRQ DF-FROM-IR D[MA] ALU[D+1] DEST[Q] COND[Q-ODD] LBJUMP[DMOVEM.MEM] $
(DMOVNX.SLO;4 2.8) {q-odd}
        DFRQ DF-FROM-IR D[MA] ALU[D+1] DEST[Q] COND[Q-ODD] LBJUMP[DMOVNM.MEM] $
(F4DEF.SLO;8 1.157) {q0-35}
        Q0-35           = 12
(F4INST.SLO;10 11.17) {q0-35}
        ALU[AC+0] DEST[D4] ENDCONN[M/D] CMUL COND[Q0-35] LBJUMP[. + 2] $
        D[AR] ALU[AC+D] DEST[D4] ENDCONN[M/D] CMUL COND[Q0-35] LBJUMP[. + 1] $
DOMUL:  ALU[AC+0] DEST[D4] ENDCONN[M/D] norm COND[Q0-35] LBJUMP[DOMUL0] $
DOMULF: ALU[AC+0] DEST[D4] ENDCONN[M/D] norm COND[Q0-35] LBJUMP[DOMUL0 + 14.] $
```