digital INTEROFFICE MEMORANDUM

TO: List

DATE: 23-June-78

FROM: T. Eggers,T. Hess

DEPT: L.C.E.G

LOC: MR1-2/E47

EXT: 6181/6448

DISTRIBUTED: 11-May-78

FILE: PBOX.SPC

SUBJ: IBOX/EBOX Functional spec.

First Printing, July 1976

0 INTRODUCTION

The purpose of this specification is to present the justification for the IBOX/EBOX multi-processor design and to describe their functional relationship to each other and memory. The Dolphin CPU will consist of two microcoded machines that will exist in a tightly coupled multi-processor environment. Each processor will have independent microcode and execution. They will not however, be capable of executing the complete PDP-10 instruction set without each other. The combination of these two processors will provide the complete instruction execution capability of the Dolphin CPU.

1.1 Description

The IBOX/EBOX tasks will be separated as follows:

1. The IBOX will handle all instruction fetches and effective address calculations.

2. The EBOX will do all remaining computation and result storing.

3. The IBOX will write no results.

The IBOX data path will consist of a 36-bit limited function ALU and the minimun number of registers needed to calculate effective addresses. In addition to these capabilities, the IBOX will be capable of completing any instruction that only affects the PC (skips, jumps, test, etc.).

1.2 Rationale

Justification for using two micro machines was made from various benchmarks designed to study instruction frequencies and instuction pair occurances. Data from these tests is available for study if desired. The most significant data point showed that approximately one-third of all instuctions executed stored no results, in either ACs or memory. The only thing affected by these instructions was the PC, therefore, a separate processor could handle this class of instruction.

The next major factor in deciding to use two processors was a program written by Mike Newman designed to study the distribution of conflicts between the results of an instruction and the effective address computation of the next instruciton. This program showed that less than 15% of all PC, index register, and indirect word fetches conflicted with the previous instructions results.

Given the preceding information, we then constructed and ran under emulation an IBOX/EBOX configuration. A test was performed on a MOVE/ADD sequence with a built in conflict. The results showed 19 clock ticks vs. 27 clock ticks on the KS10 (similar data path). This very limited test showed that indeed we could obtain substantial overlap.

The fact that the DOLPHIN cpu will be built from 1.5ns gates made it necessary to use this two processor arrangment in order to meet the desired goal of 1.5 times the KL10 speed.

1.3 IBOX Data Path

Most of the internal connections in the IBOX data path are necessary in order to compute effective addresses. In addition to this, the ability to do the functions AND and SUBTRACT in the ALU were necessary for Test and Compare instructions.

The AC Adrs Bus is used for both addressing the Ram file and microcode branching. Any of 4 items can be selected onto this bus. The indirect word register (IW) has 2-five bit fields that are used for the index register (XR) selection and the indirect bit in each the Instruction Format and Extended Format Indirect Word. The low-order bits of the Result register can be selected onto this bus for conflict compares and for fetching the effective address from the ACs. The AC field of the instruction register is also available on this bus for conflicts and AC operand fetching.

1.4 IBOX/EBOX Interface

A set of registers called "last registers" are used to comunicate results to the EBOX. They are as follows:

1. Last IR/Last AC - This register contains the opcode and AC of the instruction that has been passed to the EBOX for further processing.

2. Last E - This register contains the calculated Effective Address (EA) of the instruction that has been passed to the EBOX for further processing.

3. Last PC - This register contains the Program Counter (PC) of the instruction that has been passed to the EBOX.

These registers have a four 6-bit comparators conntected to them in order to provide the conflict logic for operand fetching. They compare the contents of the AC addrs bus with Last AC, Last AC+1, E and E+1 simultaneously. The enables for the comparators come from the Dispatch Ram and are based on what the instruction is expected to write as results. The results written classes are:

1. None - Instruction writes no results (no conflict).

2. Other - Instruction writes in unspecified locations (guaranteed conflict).

3. AC - Instruction stores in AC.

4. AC and AC+1 - Instructions stores in AC and AC+1.

5. E - Instructions stores in Effective Address.

6. E and E+1 - Instruction stores in EA and EA+1

7. Both - Instruction stores in AC and EA.

8. Both and AC+1 - Instruction stores in EA , AC and AC+1.

These registers are loaded on a "load last" signal from the IBOX microcode. An interlock mechanism on the "last" registers will prevent the IBOX from continuing until the EBOX clears this interlock, but only when the IBOX attempts to load new data in these registers. At the end of the load cycle, the EBOX will be able to dispatch on this condition (instruction available).

The EBOX will have the capability to grab and start the IBOX at any arbitrary microcode location. In addition it can pass any single data item (36-bits) in location 377(8) of the Ram File. The IBOX will have a special function to specifically read this datum.

An addtional interlock mechanism between the IBOX and the EBOX will exist for the IBOX to be able to pass data (36-bits at a time) to the EBOX via the Instruction Register (IR). The implementation of the interlock will be similiar to the "last" register mechanism.

The IBOX will also have the capability of ignoring "conflict" in order to speed up certain classes of instructions which the IBOX can fetch and decode the operands that it knows will not be written by the EBOX.

1.5 EBOX Data Path

The EBOX data path consists of a 16 word, two port register file, an adder, a shift matrix, a 256 word accumulator RAM, a byte pointer manipulator, and various multiplexors connecting these parts. The attached diagram shows the interconnections.

The data path is 36 bits wide. With the exception of floating point, all the instructions in the PDP-10 instruction set can be easily implemented using a 36-bit wide data path. The KS10 uses a 36-bit wide path and has trouble only with floating point arithmetic, particularly double precision. The 72-bit paths in the KL10 were added to make the floating point arithmetic fast. Since fast floating point is a goal only of the Dolphin floating point accelerator and not of the basic machine, the narrow data path will reduce costs without compromising system goals.

There will be no "10-bit data path" for the manipulation of floating point exponents. On the KL10, this logic was needed to make floating

point go fast. The floating point accelerator makes this logic unnecessary in the basic machine. A simpler 9-bit data path for the manipulation of byte pointers and control of the shift matrix replaces the KL10's 10-bit path.

The "magic number field" in the microcode word will be 18 bits wide. The KL10 9-bit field proved inconvieniently narrow at times, and the KS10 18-bit field proved much more useful for constants, masks, and implementing functions for infrequently used hardware.

The register file's 16 words replace the KL10's AR, ARX, BR, and BRX registers with more general purpose registers more regularly connected. The register file can be written by half-words and either half word can be written with zeros. This makes the common half word instructions trivial.

The KL10's VMA and MQ registers are combined into one VMQ register. With the register file, the only time the MQ needs to be used is with the multiply and divide algorithm steps. At these times the VMA is not active, so the functionality can be combined giving a cost savings with no performance loss.

The adder functionality is that of the 2901: add, and, xor, and ior, with either data input complemented. All of the KL10 10181 ALU's USEFUL arithmetic functions are available. All of the boolean functions are available.

The KL10's shift matrix is included in modified form: it has its first "shift by 0, 1, 2, or 3" stage built into the register file. This makes the shifting paths necessary for 2-bit at a time multiply and the times-ten path take no extra logic. The complete shift matrix will select a 36-bit window out of the 72-bit concatenated double register input. This duplicates the KL10 functionality. Slight additions will probably be made to help manipulate 9-bit characters for Cobol.

A 2-bit-at-a-time multiply will probably keep the floating point in the basic machine running at 3/4 KL10 speed.

The main data paths will have half word parity checking. Instruction retry looks too complicated, at the moment, with no apparent gain in MTBF. If we can determine that there is a large enough MTBF gain, we will examine this further.

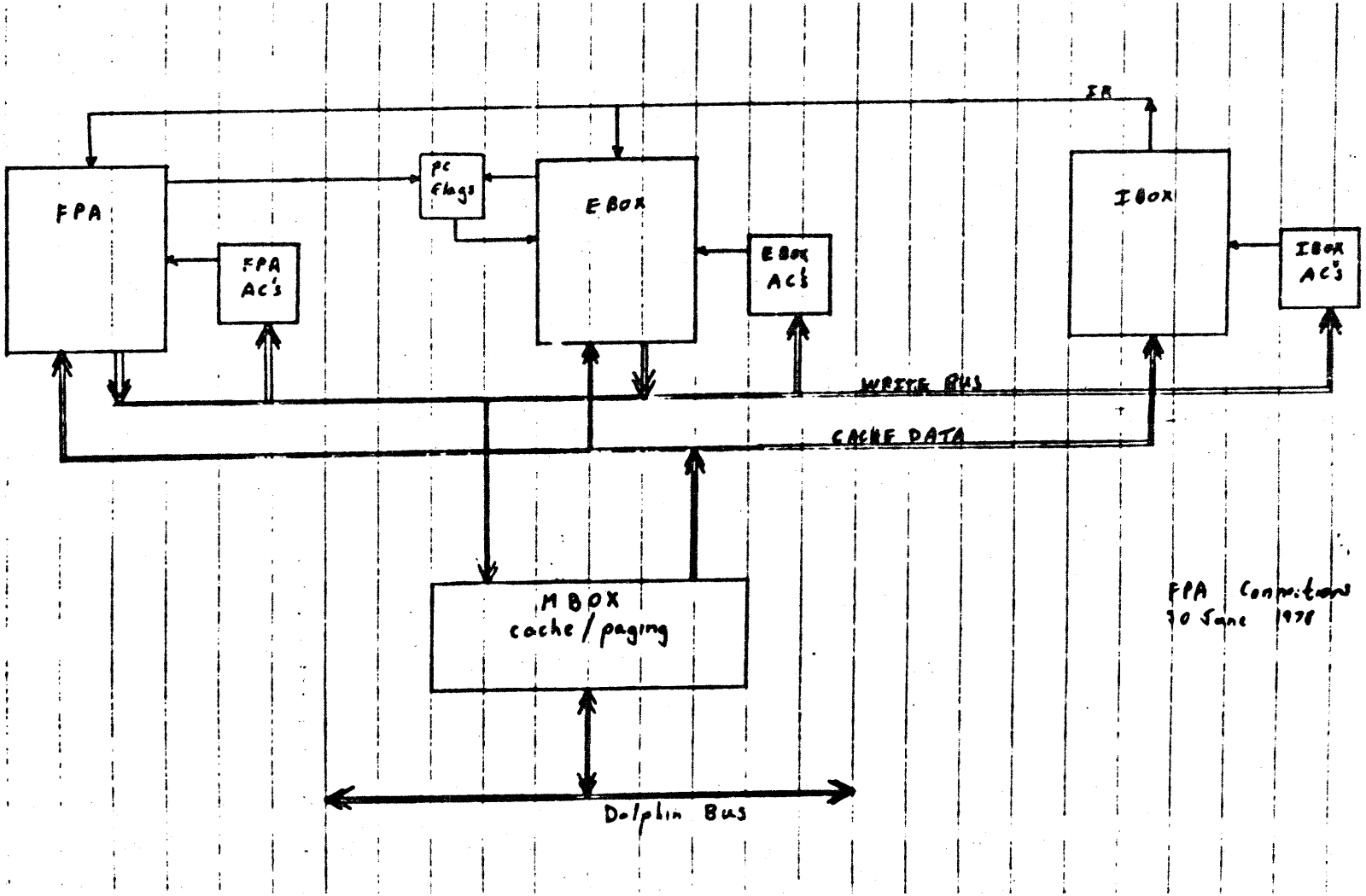
1.6 Interface To MBOX (see Don Lewine Memo Of 21-June-78)

1.7 IBOX Data Path Diagram, Attached.

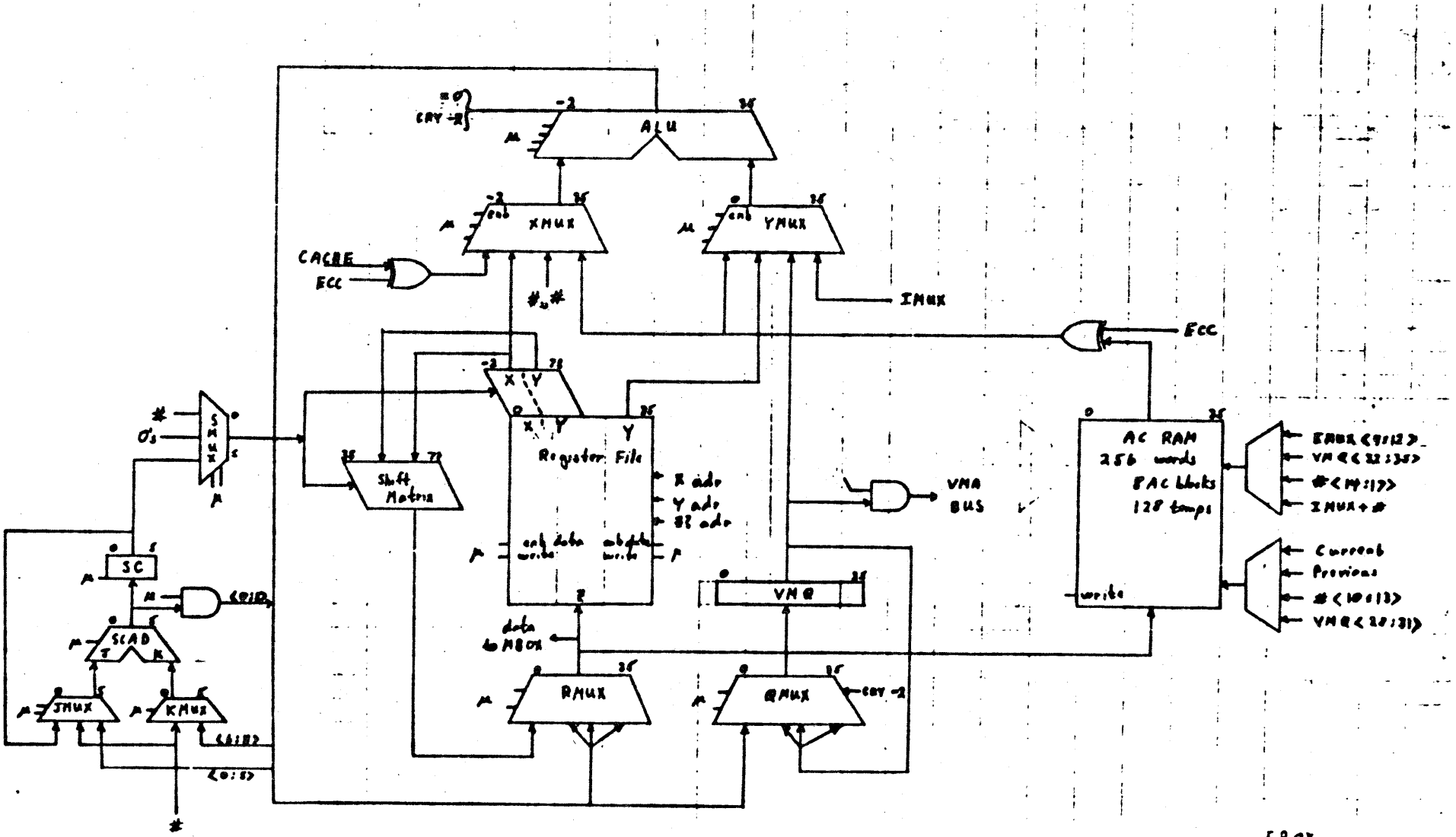
1.8 EBOX Data Path Diagram, Attached.

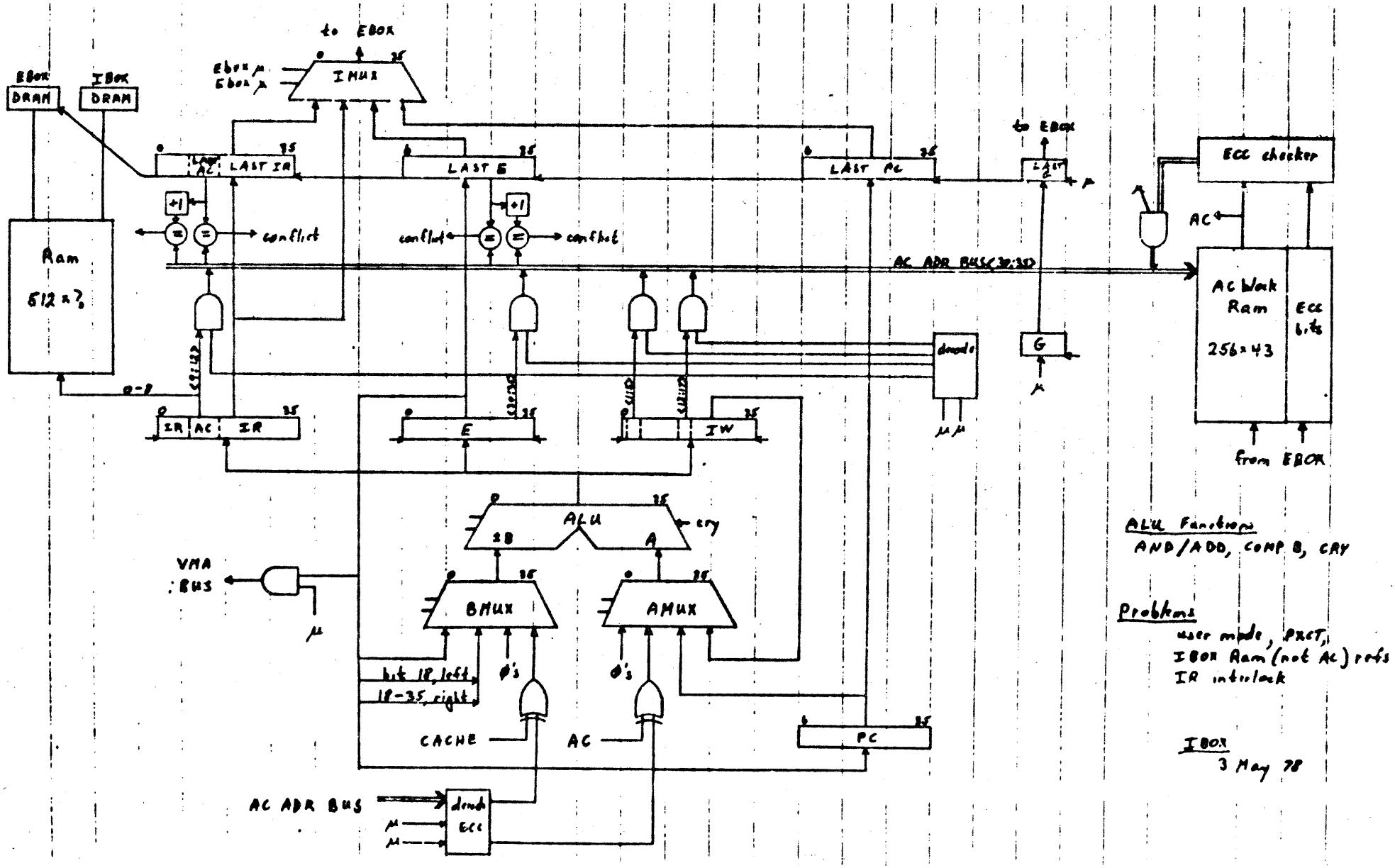
1.9 MOVE/ADD/JRST Timing Diagram, Attached.

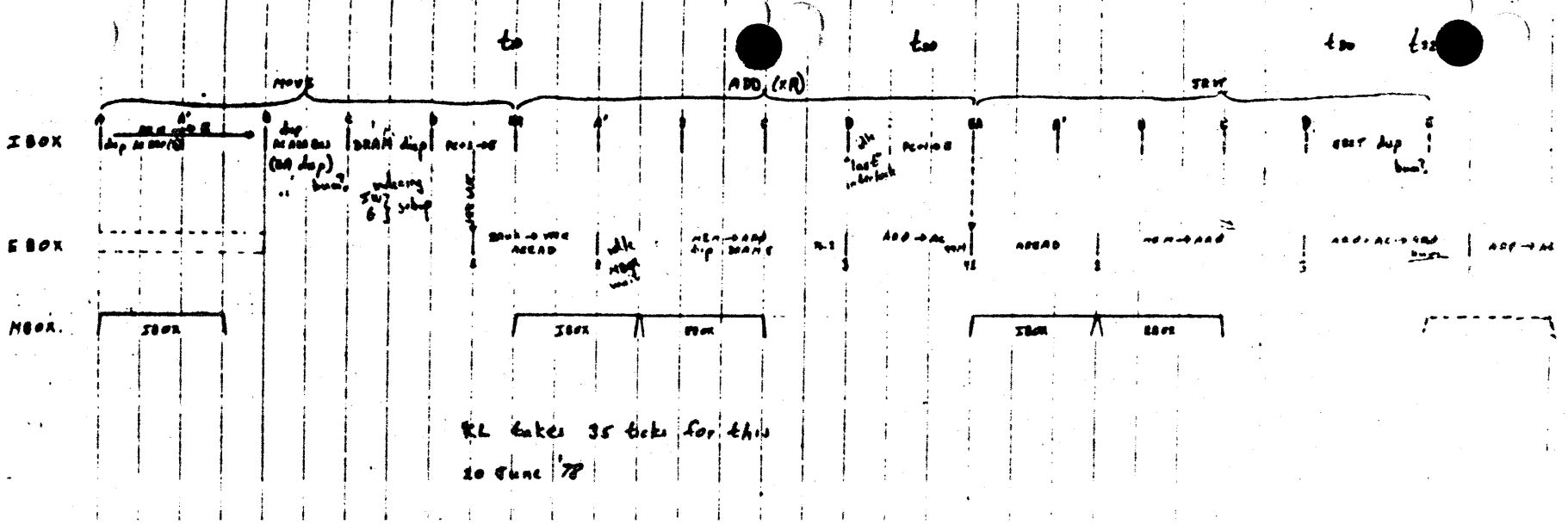
1.10 Fortran Accelerator Data Connections Diagram, Attached.



FPA Committee
20 June 1978


ALU
XMUX
YMUX
CACHE
ECC
IMUX
Ecc
X Y
Register File
Shift Matrix
SMUX
SC
SCAD
JMUX
KMUX
RMUX
QMUX
VMA
VMA BUS
AC RAM
256 words
8 AC blocks
128 temps
write
Current
Previous
EBOX
10 May 1978


to EBOX
EBOX DRAM
IBOX DRAM
EBox μ
EBox μ
IMUX
LAST AC
LAST IR
LAST E
LAST PC
to EBOX
ECC checker
+1
conflict
conflict
AC ADR BUS(30:35)
AC
Ram
512 x ?
AC Work Ram
256x43
ECC bits
decode
G
μ
μ μ
IR AC IR
E
IW
from EBOX
ALU
ALU Functions
AND/ADD, COMP B, CRY
cry
B
A
VMA BUS
μ
BMUX
AMUX
bit 18 left
18-35 right
φ's
φ's
CACHE
AC
PC
AC ADR BUS
decode ECC
μ
μ
Problems
user mode, PXCT,
IBOX Ram (not AC) refs
IR interlock
IBOX
3 May 78



KL takes 35 ticks for this

10 June '78