MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Computation Structures Group Memo 133

An Asynchronous Model of a Small Computer

by

Harold Goldberg

This work was submitted for credit in Subject 6.893, "Special Topics in the Computer Sciences" Spring 1975.

October 1975

The IBM 1130 is a small, synchronous, third-generation "midi" computer first marketed in the 1960's. It was to serve the scientific community but has been sucessfully used for commerical applications. A similar computer, th IBM 1800, has a modified interrupt structure and has been used for real-time process control. This research has attempted to develop an asynchronous model of the 1130 which uses the same data paths and internal registers of the original design, and follow the "cycle steal" and interrupt strategy dictated by IBM documents [1]. The model is based on the Kolte arrary [6, 8] implementation of Petri nets [5] a graphical representation of concurrent systems that show control dependency [2, 3, 4, 7]. We have made estimates and suggestions on the necessary size and possible internal structure of the actual array which could possibly be realized in current LSI technology.

A Basic Overview of the IBM 1130 Structure

Major Registers

The 1130 has seven major registers whose "name" and use follow:

B: MEMORY BUFFER- holds the data strobed out of memory on a read cycle or the data to be strobed in on a write cycle.

M: MEMORY ADDRESS- holds the address to be used for a memory read or write.

I: INSTRUCTION ADDRESS- holds the address of the next instruction to fetch.

D: ARITHMETIC FACTOR- holds one operand of all logical and arithmetic operations performed in the machine. It is also used as a buffer for loading other registers from memory.

A: ACCUMULATOR- holds the second operand of all logical and arithmetic operations performed in the machine as well as receiving the result of all such operations.

Q: ACCUMULATOR EXTENSION- used for double precision data manipulation in conjunction with the accumulator (A). It is also used to hold the results of a multiply operation, as well as the low order word and remainder of divide operations.

U: TEMPORARY ACCUMULATOR- as its name suggests, this register is used for holding the contents of the accumulator while it is being used in address preparation or other internal data movement.

There are six other minor registers used for holding the current instruction along with its modifiers, a carry and overflow indicator register, and a cycle control counter used for shift counting and general instruction cycle timing.

The registers are apparently of the master/slave flip-flop class in design, allowing many non-conflicting register operations to be done in parallel (including register exchanging without an intermediate buffer).

The architecture assumes three pseudo-index registers which can be used for address modification and loop control. These registers actually appear in memory as locations 1, 2, and

3, requiring extra memory cycles each time they are used. (1)

Machine and Memory Cycles

Machine and memory cycles are derived from the basic CPU clock running at 2.25 MC. There are seven major CPU cycles all of which are composed of at least eight minor T cycles, T0 through T7. (2)

There are four "I" cycles, entered for instruction fetch (I1- always entered) and address preparation where necessary. (3)

The remaining three cycles are "E" cycles that are used for instruction execution, depending on the time and number of memory references required. (4)

---

(1) The 1800 has "real" index registers within the control unit allowing faster indexed instruction execution.

(2) The term "at least" is used because in some instances T7 cycles are "extended" (repeated) to allow proper major cycle completion (add operations take a variable number of cycles, due to the interesting speed-up algorithm used). This is the first indication of the need for some asynchronous structure within the CPU. Certain interlocking strategies for interrupt and "cycle steal" control hint at others.

(3) I2 - Double word instructions, IX - Indexed instructions, and IA - Indirect addresses.

(4) Some instructions may actually complete their execution in the I cycle period - e.g. Shift instructions, since no memory refernces are required.

As may have been guessed, memory cycles are concurrent with major CPU execution cycles. A read or write may take place at T0 or T4 respectively with results expected two T cycles later. All memory reads are destructive hence all data read must be re-written to retain them in memory as well as maintain correct memory parity. (5)

Cycle Stealing

Direct memory access ability, called "cycle stealing", (6) is a feature supported by the 1130 system. A cycle stealing device requests the operation, and upon receiving an acknowledgement, places its address on the cycle steal addresss lines and puts or receives data on the I/O bus which is loaded into or derived from the B (storage buffer) register. As can be guessed, the CPU instruction sequence must be aware of the problem that the B register may be destroyed between any of the major cycles. (7) Since the system requires dealing with this

---

(5) A memory word consists of eighteen bits. Sixteen bits are used for actual data and the remaining two are used for parity check bits- one for each eight bits of data.

(6) It is interesting to take note of the term used for this operation. Apparently the system designers felt that the CPU should be in charge of memory and other memory users must "steal" cycles from it.

(7) The cycle steal address lines are fed directly to memory while the M register lines are inhibited at this time- another locking scheme is obviously needed here.

problem it dictates that no instruction cycle may execute concurrently with a cycle stealing operation. Hence there exists a second clock outlet, the X clock, which when enabled, inhibits the T clock from advancing (enabling is only done before T0 begins). Memory, however, can run on either clock.

Instruction Execution

The basic instruction cycle is as follows:

The I1 cycle loads the memory address register (M) with the contents of the instruction address register (I). An instruction word is fetched, the accumulator (A) is saved in the temporary accumulator (U), and the instruction address is incremented. Depending on various bits in the instruction, address preparation begins leading into possible I2, IX, and IA cycles. The cycle control counter is loaded appropriately and actual execution of the instruction begins. Execution of the instruction causes the CPU to enter necessary E1, E2, and E3 cycles finally leaving the cycle control counter zero and when the next T0 begins a new I1 cycle is entered.

Interrupts

By design, interrupts can only occur before an instruction begins execution. Due to the implementation memory

cycles are inhibited at the time of the interrupt resulting in another seemingly asynchronous process. (8) A "Branch and Store Instruction Address" instruction is forced to be executed, causing the execution point at the time of the interrupt to be saved.

There are six interrupt levels (0 through 5) of decreasing priority (5 lowest). Each level may interrupt any lower level. Six words of memory are reserved for the "interrupt vectors" which are addresses of routines to be entered upon detection of that level interrupt. Due to the nature of the "Branch and Store Instruction Address" instruction, as well as the lack of interrupt masking, (9) re-entrant procedures are difficult if not impossible to write.

---

(8) The I/O bus is used to gate the proper instruction for execution into the B register.

(9) The 1800 has this feature.

The Model

Petri nets and the Kolte Array

Petri nets consist of places and transitions connected by directed arcs. These arcs are only allowed to connect dissimilar items (no arc connects a place to another place or a transition to another transition). In the class of nets modeled by the Kolte array, (10) a place may either have a token or be empty. Places which have arcs originating at them and terminating at a transition are known as input places to the transition. Places having arcs directed at them from transitions are called output places of the corresponding transitions. A transition is said to be firable when all of its input places have tokens. The firing procedure entails first removing tokens from all of the input places of the transition to be fired, then placing tokens in all of its output places.

Two or more transitions are said to be in conflict when they share at least one input place. This obviously becomes important when both transitions are firable at the same time. In such situations an arbiter of some kind is used to choose the proper transition to fire. (11)

---

(10) The class modeled is all safe nets. What this means is that at any time we may only have one or no token in any given place at any time.

Petri nets can easily be represented in matrix form (12) with rows representing transitions and columns representing the places. At the intersection of rows and columns appropriate marks are placed to indicate the interconnection of the places and transitions. A dot (.) is used to represent an input place to a transition, while a cross (x) is used to represent an output place of a transition. Patil [6] extended the notation to allow binary testing of places to enable transition firings. One could test whether a place had a token (1) or not (0) without actually using up the token when firing. This extension allows the testing of logic levels that we may introduce to our net. We extend the notation to allow setting of DC logic levels with the SET (s) and RESET (r) positions.

The Kolte array is the Petri net matrix realized. Even the new additions are easily implementable. The array basically consists of a diode matrix with appropriate terminating circuitry for simulating operation of places and transitions. All transition circuits are similar, but there are now four place circuits: input places, output places, internal places, and DC output places. We do not intend to give a full explanation of

---

(11) Since we are dealing with safe nets, one of the conflicting transitions usually disables all others from immediately firing due to the removal of the input tokens at the time of firing.

(12) Patil [6] suggests that it was Holt who first used the matrix notation to represent large Petri nets on small pieces of graph paper.

the operation of the array but suggest that the unfamiliar reader see [6, 8] for details. We believe that the extended Petri net matrix is fully realizeable, operating in parallel fashion, in the revised Kolte array. Our DC place extensions are realizeable using the existing input place circuit, and our own output place circuit shown in figure 1.

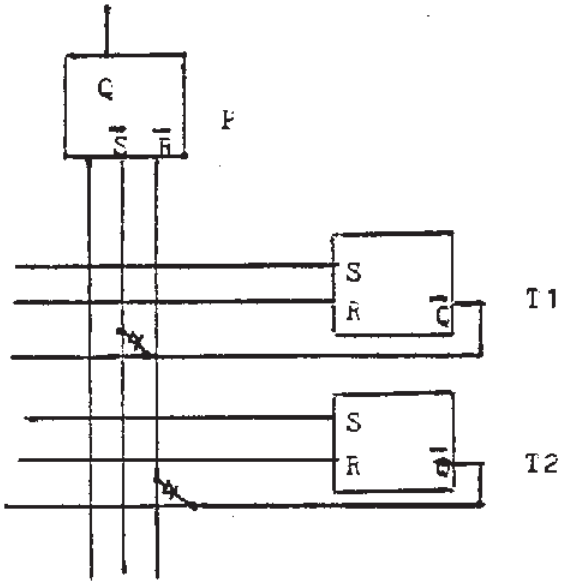


Figure 1. Implementation of the DC output place.
Here T1 can "set" P, while T2 can "reset" P.

We do not allow using any output places for input nor input places for output.

The use of the DC places is clear in conditional branching applications just as Patil has demonstrated in [6]. However we do not wish to be bothered with using up tokens when presented, nor clearing only those places where we have put

tokens for output. We are not forced to reset tested bit positions when we are done with them. In using these places we must trust that the source of the DC level has set it accordingly and send acknowledge signals back only after these signals are made availble. Similarly we transmit these signals, possibly status or results of intenal operations, (13) by use of these DC output places. We must believe that the acknowledge signals arrive after, or at worst the same time as, the DC levels that they apply to do. This can be insured by appropriate cabling (use same cable paths) or appropriate delay module insertions.

Details

We have attempted to produce a faithful model of the internal control of the 1130 but must obviously stray from an exact modeling for certain reasons. We wanted to develop an asynchronous model hence clock pulse dependencies have been left out. We have allowed instruction execution to proceed during cycle steal operations up to a point where they would reference a possibly modified value (the B register). We did not attempt to model immediate stop, single cycle execution, single step execution, or other hardware debugging controls (14) (although

---

(13) We are assuming multiple array chips with interconnections in all of this discussion.

(14) Patil's [6] use of a programmable place would allow stops to be inserted at any point, however.

single I cycle execution would not have been too difficult with the current model). Our main purpose was to determine the approximate size of an array necessary to perform the functions of the CPU.

Register/Arithmetic Unit

For purposes of the model we have assumed that there exists a register and arithmetic unit whose capabilities follow:

The unit contains all the major and minor registers of the 1130. It provides parallel register transfer operations (those allowable by the 1130 design), register reset, and certain arithmetic and logic capabilities. We are not saying that we have a full ALU, but we do have the ability to perform six operations specifically between the D and A registers, leaving the result in the A:

(1) transfer

(2) logical ANDing

(3) logical ORing

(4) logical EXCLUSIVE ORing

(5) ADDing (resulting in a possible "temporary carry")

(6) SUBTRACTing (resulting in a possible "temporary borrow")

We have not really modified the system structure by this grouping and in fact not changed any controls by including the ADD/SUBTRACT logic in the unit. We have simply centralized the point where the "set arith logic", "perform add/substract cycles" loop, and "reset arith logic" functions are done. (15) We similarly allow the register/arithmetic unit to handle left and right shifts of the A, and the A and Q combined.

When returned to with the "register operations complete" token we can assume that tests on DC levels like "temp carry/borrow", A bit 0, A bit 15, Q bit 0, Q bit 15, etc. are valid.

Use of this unit is accomplished by setting appropriate DC output places to one to indicate the desired operation and then placing a token in "register do". We assume that upon receiving "register operations complete", all the DC control lines are reset (possibly by a transition that has a field full of resets (r's) for all those places. Some operations are binary coded to add density to the control lines, but this technique has not been fully taken advantage of. (16) We have taken the

---

(15) These functions are always done whenever an add or subtract operation is done within the CPU.

(16) To do so would require a study of which functions could never be done in parallel, and then assign a unique binary quantity to all other possible combinations of functions.

liberty to allow a control line that moves the TAG register to the M (the TAG register holds the index register number which is equal to its storage address). In the real i130 this is done by inhibiting M register output and ORing the TAG lines onto the memory bus. We have not specified how this is done in out model and in fact we could relabel that control line to "TAG to memory address bus" to be correct.

Memory Control

We have developed an interesting priority memory control scheme for use by our cpu modules:

Each module has five control lines for memory control....

(1) Read
(2) Read Acknowledge
(3) Write
(4) Waiting Write Acknowledge
(5) Write Acknowledge

We dictate use of memory by assigning the Write Acknowledge token the job of being the memory locking "semaphore". Once a module has acquired the Write Acknowledge token it may issue a read and subsequently a write request without worrying about other module interfernce. The key to the success of the inter-module control is in the Waiting Write Acknowledge line which gets carefully

arbitrated by the Memory Interface Module (MIM) - a module specifically designed for this task. Hence inter-module communication about memory use is essentially eliminated and the task of supporting cycle stealing is easily accomplished by correct arbitration by the MIM. A pictorial diagram of two competing modules connected to the MIM is shown in figure 2.

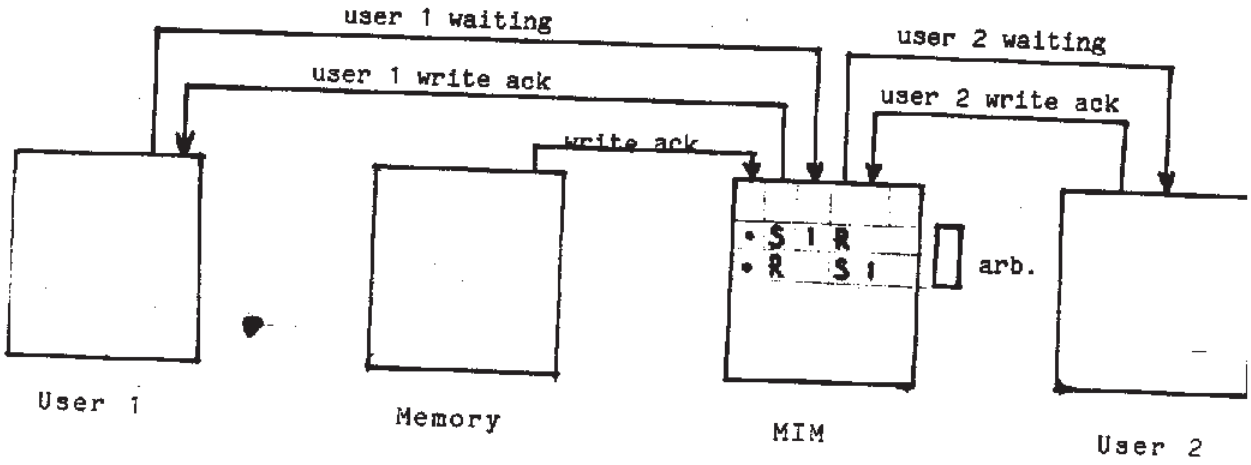


Figure 2. The write acknowledge interlock scheme.

Read and Read Acknowledge is simply handled by the MIM by its noting who asked for the request (only one module could legally ask at any time) and sending him the acknowledge upon receiving it from memory.

We again point out that all reads are destructive, hence the Write Acknowledge scheme is sufficient since once a read has been done by a module, it is committed to doing a write (with possibly changed data). (17)

---

(17) In modern systems memory handles write-backs internally and thus has functions such as read only, write only, and read alter write, in which case extended arbitration is necessary.

Results

We must first point out that a complete model was not designed due to time considerations. We have however looked ahead, and tried to supply enough information in the form of control lines and DC levels to support all unimplemented instructions and functions of the CPU that we have not modeled. By specifying memory interface controls, we have essentially dealt with the cycle stealing aspect. The interrupt policy is also easily dealt with by our standard use of the "NEXT I1" place (18) which can be arbitrated in favor of the interrupt logic control module.

We have a complete and detailed model for the full instruction fetch, decode, and address preparation cycles, all in one module. (19) We have also modeled seven instructions completely (20) which when viewed in conjunction with the aforementioned I cycle module, exercise almost all lines of our register/arithmetic unit.

We have presumably accomplished our goal in determining

---

(18) Signifying that the next instruction should begin. This is the time that an interrupt may occur in the real 1130.

(19) Our module size was chosen solely by available graph paper size, but proved to be excellent do to the grouping of all the I cycles in one module.

(20) At the time of the writing of this paper.

the approximate size of the required array. We have found that the major potion of the array places were consumed by DC input and output places. (21) This is so because of Patil's [8] suggestion of breaking up and staggering place columns. The idea here is to break up one column (a single place) into many smaller places. This is done to a few columns which are then vertically arranged so that the new places overlap. Patil had suggested using small places of length three. We have found these to be adequate for sequential control and possible double "branching" but have also found the need for longer places (though not as long as the entire column). We have chosen places of length eight in addition to those of length three, and overlapped these places as well. (22) The use of these longer places was found to be low in the instruction execution module but proved to be invaluable in the I cycle module. In fact this technique had cut down the number of internal place columns from about fifty for the I1 cycle alone, to about six for <u>all</u> I cycles together- a marked improvement. For this reason we were not very careful about how many full length places we had used but found that we only had used six, and even ther use might have been avoided. Of course there exists disadvantages with using these "close-packed"

---

(21) An expected result do to the "place multiplexing" scheme described further on.

(22) Our choice was made by examining an initial version of the model and determining what size place could best suit its needs. Eight was chosen but is by no means the only size that would have worked.

places. Patil has described the use of position independence in overcoming chip defects, and control flow modification. It is these problems which concern us now. We have found that we can still gain some of the independence by using enough free long columns and splicing in "branches" to other parts of the module with subsequent "returns". (23) We can even maintain the "immediate stop" capability by utilizing Patil's scheme of a programmable place: In this place we could modify any transition to have it as an input place. Then with appropriate setting or resetting this place we have the programmable "breakpoint" effect.

We have found need for about twenty input places and approximately forty-five output places (many for the register/arithmetic unit). This is obviously high compared to the number of pins we find on conventional LSI chips today (about 40). However, suitable techniques for splitting transitions can, and have been found. In fact with proper control organization, modules can be designed to only contain a subset of input and output places, having only those transitions which make use of these lines present. This of course leads to a tradeoff that we have not studied: input and output places for the system as a whole, versus those used solely for inter-module communication due to module separation.

---

(23) Note the similarity between this technique and "patching" program object code.

We should mention that in our model the largest size arbiter required was a three-input arbiter. We believe this was mainly do to the addition of binary testing to the array. In fact, in many cases, abiters were eliminated entirely. We found them to be most useful in the "IF (SPECIFIC TEST) THEN DO ..... ELSE DO ....." case, and for true priority queing.

Summary

We have found that our model was not difficult to construct due to the availability of appropriate documents. We saw how the 1130 atempted parallel wait functions by clock manipulations and inhibit lines.

We believe that with appropriate modularization of the CPU functions, the model could in fact be realizeable.

It was was learned that the "close-packed" places could be enourmously helpful in cutting down the total number of internal place columns used, hence reducing the overall width of the array. It was shown that the size primarily depended on input and output connections, a seemingly universal result when one attempts to modularize a system. We also saw how the use of binary testing reduced the size and number of arbiters needed. Our model required approximately 80 place columns in all: Approximately 20 input, 45 output, 8 full length internal, and 6 staggered internal place columns were used.

We believe that in using the array one can reap all the advantages of micro-programming, yet have the system run at top speed, using speed independent modules.

References

[1] International Business Machines, Field Engineering Manual for the IBM 1131 CPU, volume 5.

[2] Hack, M. H., Analysis of Production Schemata by Petri Nets, Technical Report TR-94, Project MAC, M.I.T., February 1972.

[3] Holt, A. W., Final Report of the Infomation System Theory Project, Technical Report RADC-TR-68-305, Rome Air Development Center, Griffis Air Force Base, New York, 1968.

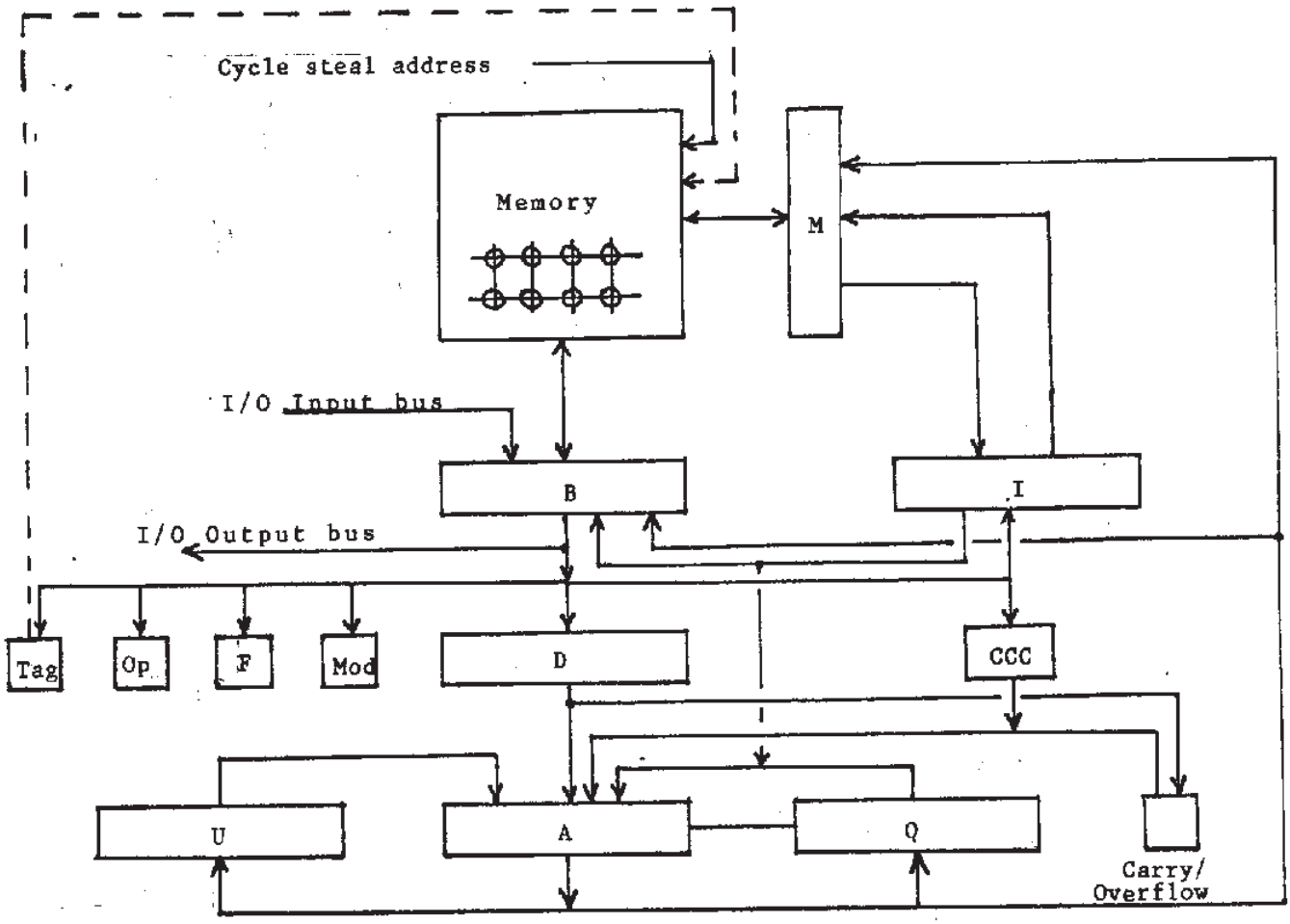
[4] Holt, A. W. and F. Commoner, Events and Conditions, Record of the Project MAC Conference on Concurrent Systems and Parallel Computation ACM, New York 1970, 3-52.

[5] Petri, C. A., Communication with Automata, Supplement 1 to Technical Report RADC-TR-65-377, Vol. 1, Griffis Air Force Base, New York, 1966. Originally published in German: Kommunikation mit Automaten, University of Bonn, 1962.

[6] Patil, S. S., An Asynchronous Logic Array, Technical memo TM 62, Project MAC, M.I.T., Cambridge, Mass., May 1975.

[7] Patil, S. S., Coordination of Asynchrounous Events, Technical Report TR-72, Project MAC, M.I.T., Cambridge, Mass., June 1970.

[8] Patil, S. S., Micro- Control for Parallel Asynchronous Computers, Proceedings of the Euromicro Workshop, Nice, France, North-Holland Publishing Co., 1975.



IBM 1130 Internal Data Paths