bcc	title	prefix/class-number.revision
	THE REMOTE CONCENTRATOR DESIGN	RC/S-23

checked	authors	approval date	revision date
Robert W Jones	Paul Heckel	12/17/69	
checked		classification	
C. Machen		specification	
approved		distribution	pages
Mel		company private	32

ABSTRACT and CONTENTS

This document describes the Remote Concentrator for the Phase Two CHIO. It includes a description of the algorithms the Remote Concentrator uses and the opcodes for the Processing Unit in the Remote Concentrator.

The interface with the communications system (high speed line to the CHIO) is not described however.

Contents

Design of the Remote Concentrator

The Remote Concentrator is a 16 bit microprocessor that is connection to a 16 bit memory. It runs several tasks:

1,2,3) High Speed line interfaces
4) Model 37 teletype
5) Model 35/33 teletype
6) IBM Selectric
7) Teletype special condition task
8...n) Special devices (line printer, card reader, etc.)

The maximum number of tasks that it handles is 32.

The Remote Concentrator normally is executing one of these tasks when an interrupt comes in. When this happens the two words of interrupt bits are merged into a two word Waiting Task Word (WTW). A bit is set in the WTW for each task requiring attention. If the running task is an interruptable one and it is of a lower priority than the new task, the task being executed is stopped and its state is saved. There is a task table which is indexed by the task number. It gives the location of the subroutine to be called when the task is activated. It also keeps the state of the task. This table has two bits, one that says the task is micro-coded, and a second that says the task is non interruptable. The state of the new task is loaded and it is called. The new task will run until it either blocks or it is interrupted (if interruptable) by a higher priority

task.

When a process blocks, the bit in the Waiting Task Word associated with the process is turned off, the state is stored, and the task of highest priority according to the Waiting Task Word is called in the same manner that this task is called.

The above method of dispatching on a task is reasonably efficient, and it allows a microcoded process to be replaced by an RPU process. However, it allows an infinitely looping program to prevent any lower priority task from being executed. This is unfortunate but the number of such un-debugged tasks should be quite small. Initially such a task can be run as a low priority task if there is the possibility of this error. There are other possible errors as well. The system is neither fool proof nor knave proof. However, a good programmer has a minimal chance of affecting other tasks with his errors.

Getting the RC out of an infinite loop in a non-interruptable task requires manual intervention.

The CHIO Interface Task (CHIT)

The task that interfaces with the 4800 baud line performs a rather complicated function with two possible end results. It either gets a character to be placed into a specified line in the Remote Concentrator; or it must get a character from an available line in the Remote Concentrator to send to the CHIO. It will get or receive these characters from the CHIO in a manner to be described in a forthcoming document. The details are not necessary here.

If this task has a character for the nth line it will index into the local line table (see appendix II for table format). If the specified device is a teletype or 2741 it will store the character in the appropriate field and set a bit saying that the character is there. (The CHIO will not send characters to a device faster than the device can accept them.)

If the device is a high speed device the table entry contains a ring buffer descriptor that specifies a ring buffer to store the character into (see the SBRB instruction). The process associated with this task is awakened by setting the bit in the WTW.

If an input word is needed for the CHIO, CHIT finds the next line with a character for the CHIO by linearly searching the local device table (starting with the line following

the line where the last character was found). It will get one character from this line and output it. When next called it will start with the next line to look for a character.

There is one CHIO interface task for each high speed line to the CHIO. There may be as many as three such lines, therefore as many as three such tasks.

High Speed Input Device Tasks

If there is input from a high speed device this will be reported by the appropriate interrupt bits being set. The task associated with this is called to service the device. It will read characters from the device and should write them in the ring buffer associated with the line. This task will continue to run until it blocks. (It may be interrupted temporarily to run a higher priority task).

The CHIO interface task will read the characters from that line and send them to the CHIO.

High Speed Output Device Tasks

Whenever a character is put in a high speed output line ring buffer, a bit is set in the Waiting Task Word, causing the output task to be called. This task will read characters from the buffer and output them on the specified device.

To give an interesting example of how this might work, consider interfacing with a line printer. Assume the printer task cannot output characters to the printer until it has received a whole line image, and when it has the whole line image it must output the characters at a rate of one character every 500 microseconds. This task will get awakened each time a new character is placed in its buffer. It can then check the character to see if it is a carriage return. When it receives this it goes into another mode outputing a character on the line printer and then blocking. When the printer is ready for another character it sets an interrupt which reactivates the process. In actual practice a simpler algorithm could be used; but this seems to be a good example showing how output from a high speed device can work.

Teletype Interrupt Task (TINT)

The teletype interrupt task for the remote concentrator is similar to the subroutines that handle the teletypes on the Phase 1 CHIO. There is one primary difference. For each low speed device there is an oscillator running at seven times the bit rate for the device. Each (oscillator) cycle an oscillator will trigger the interrupt associated with the device. This will cause the specified teletype task to be called.

When the Teletype Interrupt Task (TINT) is called, it will scan the bits for an input character in the same way that the Phase 1 CHIO does its bit scanning. (see IHTWD/S-). When it has a character to be input, it stores it in the local line table and sets IHC. (see Appendix II). CHIT will come along, and finding the bit set will send the character to the CHIO. Similarly character output is the same as in the Phase 1 CHIO.

There are several tasks related to teletypes, that the CHIO must perform that it does not do in the Phase 1 CHIO; but all of these are done by the Abnormal Teletype Condition Task.

Abnormal Teletype Condition Task (ATCT)

There is one task which is called whenever an abnormal teletype condition occurs. Examples of this are a carrier going off or a line's needing to be dialed. If any data set detects an abnormal condition it sets the appropriate interrupt so that the abnormal teletype condition task will be called. This task will then execute the appropriate PINs to find out what the problem is. It will then send an indication of the problem to the CPU.

One of the local lines, called the input control line, is reserved for control information from the CHIO to the remote concentrator. If any characters are put in this buffer the Abnormal Teletype Condition Task is also called. One of its functions is to dial a number of the selected line. (see page 137)

The Abnormal Teletype Condition Task, on a command from the input control line, will hang up a teletype line.

When a call comes in on a (n unused) line the Abnormal Teletype Condition Task will send the information to the CHIO (and a CPU program) on the output control line indicating this. The CHIO will then set the line to some new value by sending another request to the ATCT setting the device type. ATCT also handles line recognition.

Line Recognition

All devices will come in on the same telephone lines. The algorithm for determining the device type is implemented with a CPU program.

1) When a line into the Remote Concentrator is called the Abnormal Teletype Condition task will send the line number to the CPU.

2) The CPU program will send a request to the Remote Concentrator asking it to answer the line, set the unknown device type bit, and set the NCIP bit in each of the device entries for the specified teletype.

3) The bit scanning subroutine will independently examine the input stream as if it came from three different devices. When the bit scanning subroutine receives a character it notes that the unknown device type bit is set, and therefore will not send the character in on the input line as would be the normal case, but it will send the character in on the input control line with an indication of the device type assumed and the line number.

4) The user (because he has been brainwashed by marketing) will type a carriage return.

5) The CPU program will receive several messages from the

input control line, one of which will say that a device of a specified type has input the character Carriage Return. The CPU will then know what the device type is, and set the device type correctly in the Remote Concentrator.

6) If the Device is a 2741 the CPU program will type out either:

TYPE %

@IDU :

or:

ULA: &

TYPE %

The user, mystified by one of these messages, will hopefully type a percent sign, in response to the comprehensible half to identify the device type. The CPU might have to reassign the device type as there are two completely different 2741 character sets and they both have the same code for carriage return.

The user knows that to get into the system he calls a certain number and then hits the carriage return once or twice. He should then find that the computer recognizes his device and types out a message asking him to log onto the system. Only in the case of the 2741 is any further typing necessary.

The discipline proposed has been designed to be reasonably general, allowing the algorithm to be changed. All of the high level decisions are made by the CPU program.

Character Echoing and Line Resynchronization

If a line is in local echo mode when the CHIO Interface Task takes a character from the line to send to the CHIO the CHIO Interface Task will determine if the character should be echoed, and echo it if needed. It will also reset Local Echo Mode if the character is a break character.

The CHIO can reset local echo mode by sending a resume echo character to the appropriate line. Whenever a Character is sent to the CHIO from the Remote Concentrator, LVB is set to N (a specific constant). Whenever a new block comes from the CHIO, LVB is decremented by 1 until it reaches zero. If LVB is zero when the Resume Echo Character is received, LEM is set. The reason that this algorithm works will be described in the forthcoming document on CHIO/ Remote Concentrator Communications.

A precise description of the algorithm for local echoing of characters (which is identical to the Phase 1 system) follows:

1) Send the character to the CHIO. If LEM is off terminate, and reset LVB to N; Terminate
2) If the character is a non echoable character, terminate
3) Otherwise, get the character type from the Character Type Table (same as in Phase 1) and

add the type to the break strategy. If the result is less than 4 echo the character and terminate

4) Otherwise, turn off LEM.

5) If the EBC bit is on, echo the character

Because the CHIO has the same information it will know precisely what happened.

The procedure just described is performed by a RPU program. Called by the CHIT its calling address is taken from the character type table. It could be replaced with a different subroutine if some other echoing strategy is required.

Loading and Storing Programs in the Remote Concentrator

Programs and data can be loaded into the remote concentrator in three different ways. There is a control character which if seen by the communications system will cause the Remote Concentrator to be loaded (see forthcoming document). This is normally used for Initial Program load. Second, one of the ring buffer descriptors can be used to point to the palace in core that a program should be loaded, and characters can be sent to the Pseudo device associated with this buffer.

The third method is to send a request to the Abnormal Teletype Task to load a field or Ring Buffer Descriptor. The section on Control Line Messages will give details of this method. All of these methods can be used for either loading the Remote Concentrator or fetching its contents so that a CPU program can look at it.

Messages sent to the CHIO on the Input Control Line

The Abnormal Teletype Condition Task (and possibly other tasks) may send information to the CHIO on the Input Control line. A CPU process associated with the line will process the message.

Messages are sent in the M1 character set. (This is a matter of convenience as it allows me to say that a message beginning with the character A means .. rather than a message beginning with a 40B means.)

Some messages are followed by a character which is interpreted as a line number. This means that the character following is equal to 40B plus the line number. (a max of 224 lines 112 input and 112 output.)

The messages that can be input to the CHIO/CPU are:

1) A line; The specified line is ringing.
2) B line; The specified line dropped its carrier
3) C line; The specified line's carrier on signal turned on.
4) D line; The specified line hung up;
5) E line char; The specified line is in an unknown device type mode. The Remote Concentrator, assuming it was a model 37, interpreted the specified character as the character that was input.
6) F line char; Same as E except a Model 35 is assumed.

7) G line Char; Same as E except that a 2741 is assumed.

8) H line; The specified line answered the phone (the computer calling out)

9) I line field VAL1 VAL2; Value is the value of the specified field for the specified line.

10) J scratch VAL1 VAL2; specifies the value of a scratchpad register.

Similarly messages can be directed to the Abnormal Teletype Condition Task by sending a message of the same form to the output control line. The specified messages are:

1) A line; Answer the phone on the specified line

2) B line; Hang up the phone on the specified line

3) C line NØ N1 N2 N3 N4 N5; the automatic dialing device will interpret the seven low order bits as a two digit number, for N1 to N5. This specifies a ten digit number of dial. NØ is the number of digits to be sent to the automatic Dialer.

4) D line; Put the specified line in unknown device type mode.

5) E field line, VAL1 VAL2; Index into a field table to get a field descriptor and store the specified value into the specified line.

6) F field line; Send the value of the specified field to the CHIO/CPU.

7) G line d1...d8; The descriptor in the line is loaded with the 4 words specified by the 8 characters. Note that if the specified line is an input line the Remote Concentrator will start

sending characters specified by the line to the CHIO (the buffer is not empty).

8) H scratch VAL1 VAL2, the specified scratchpad is loaded

9) I scratch; Return value of specified scratchpad register.

Communications with the CHIO

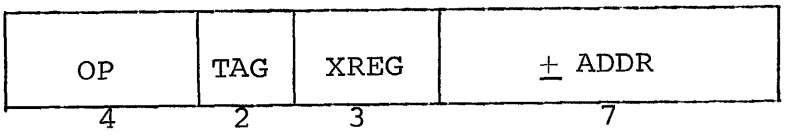
How does the Remote Concentrator communicate with the CHIO? How does it find and correct errors? How does it encode the lines that the characters are sent to? How does resynching for Echo work? Tune in to next week's exciting document and find out.

APPENDIX I

THE REMOTE PROCESSOR UNIT

Instruction Format

Each instruction consists of 16 bits divided into four fields, the opcode field, the tag field, the index register field and the address field:



The first stage of the address computation is the adding of the 16 bit byte address specified by XREG to a signed 7 bit word address in the instruction (ADDR) shifted left one to form a 16 bit Preliminary Address. This may be interpreted as a byte address, or the left 15 bits may be interpreted as a word address.

The tag field specified 4 types of addressing: indirect, direct, immediate, and scratchpad.

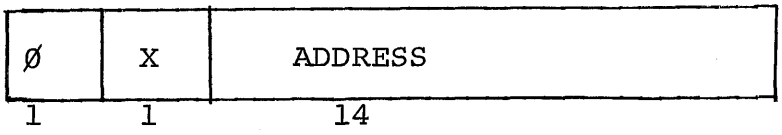
Direct Addressing (0): The Preliminary Address is the Effective Address of the instruction.

Immediate Addressing (1): The Preliminary Address is treated as the contents of the Effective Address of the instruction.

Scratchpad Addressing (2): The word part of the Prelinary Address is the scratch pad number being addressed.

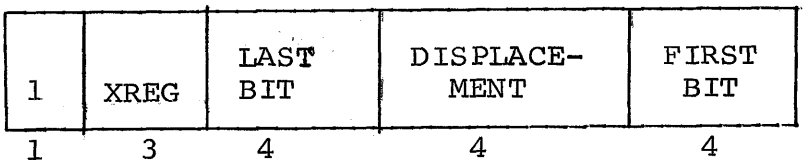
Indirect Addressing (3): The word part of the Preliminary Address is the address of an indirect word.

An indirect word may be of the following format:



The first bit must be a zero, the second bit maybe a one to specify indexing by the index register #1. The address field specifies a 14 bit byte address.

Alternatively the format may be as follows to specify a field:



The displacement (4 bits, no sign) is added to the specified index register to specify the word to be addressed. The beginning and end bits of the address field is specified by the first bit and last bit fields in the indirect word.

Machine Registers

This machine has 8 addressable index registers, which constitute the state of a RPU process.

0) Z register always contains zero
1) Index register
2) Program counter
3) Accumulator
4) Link
5) XR5
6) XR6
7) XR7

The addresses contained in these registers are byte addresses. Since the address field of an instruction is a word address, it is shifted left one to add it into the index register to compute the effective address of an instruction.

Machine Instructions

LXR (1): Load index register

BRY (2): Branch (load program counter)

LAC (3): Load accumulator

LRL (4): Load return link

LR5 (5): Load LR5

LR6 (6): Load LR6

LR7 (7): Load LR7

The specified register is loaded with the contents of the effective address of the instruction. The low order (byte) bit of the effective address is ignored.

STA (10): Store

The contents of the accumulator is put into location specified by the effective address. The low order (byte) bit of the effective address is ignored.

LDB (11): Load byte.

The addressed byte is loaded into the right half of the accumulator, the left half of the accumulator is cleared.

STB (12): Store byte.

The right half of the accumulator is stored in the byte addressed by the effective address.

ADM (13): Add to Memory.

The contents of the accumulator is added to the specified memory location. (The low order (byte) bit in the effective address for this instruction is ignored.)

ISZ (14): Increment and skip if zero.

The contents of the effective address is increased by 1 and if the result is zero the next instruction is skipped.

BSL (15): Branch and Save Link.

The location counter plus 1 is put into the Return Link (XR4) and the next instruction is taken from the location specified by the effective address.

STL (16): Store Link.

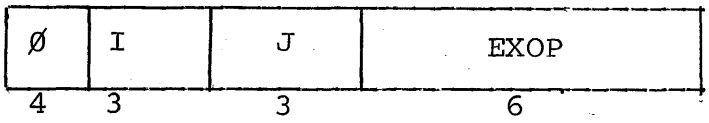
The link register is stored in the location specified by the effective address.

Spare (17): Not yet defined.

EXT (Ø): Extended opcode Set.

The Extended Opcodes

The opcodes in the extended set are of the form:



The effective address of extended opcodes in ignored. Thus there are 64 possible extended opcodes, each of which specifies 2 registers: an I register and a J register.

ADD (Ø):
The contents of the I register is added to the contents of the J register.

SUB (1):
The contents of the I register is subtracted from the contents of the J register.

XCH (2):
The contents of the I register is swapped with the contents of the J register.

COPY (3):
The contents of the I register is copied into the J register.

OR Logical Or (4):
The contents of the I register is Logically Ored into the J register

EOR Exclusive or (6):
The contents of the I register is Exclusively Ored into the J register

SKE (7), SKNE (10), SKG (11), SKNG (12), SKL (13), SKNL (14): these six instructions will compare the I register with the J register and skip if the I register is equal to, not equal to, greater than, not greater than, less than, or not less than the J register. Neither register is changed.

SKM (15), SKNM (16): Skip Masked, Skip Not Masked
SKM will logically AND the I register with the J register and skip of the result is not zero. SKNM is identical except that it will skip if the result is zero. Neither register is changed.

ISG (17): Increment and skip if greater
The I register is incremented by 1 and then compared with the J register. If the new I register is greater than the J register the instruction skips.

DSL (20): Decrement and Skip if Less
The I register is decremented by 1 and the instruction skips if the new I register is less than the J register.

IWSL (21): Increment word and skip if greater
The I register is incremented by two and if the result is less than the J register it will skip. The low order (byte) bits are assumed zero in this comparison.

DWSKL (22): Decrement word and skip if less

The I register is decremented by 2 and if the result is less than the J register it will skip. The low order byte bits are assumed zero for the comparison.

SWB (23): Swap Bytes.

The left byte of the I register is swapped with the right byte of the J register.

LLB (24): Load Left Byte

The left byte of the J register is put into the right hand byte of the I register, the high order bits being cleared.

LRB (25): Load Right Byte

The right byte of the J register is loaded into the right byte of the I register, the high order bits being cleared.

BLK (26): Block

Control goes back to the microprocessor which saves the state. When the microprocessor resumes control it will commence at the next location.

LBRB (27): Load Byte from Ring Buffer

The I register specifies a pointer to a ring buffer descriptor (diagram in Appendix II). A ring buffer descriptor is a 4 word entry each entry being a byte pointer, the first word points to the beginning of the buffer, and the last

to the end of the ring buffer. The second word points to the read pointer, and the third word to the write pointer.

If the read pointer is the same as the write pointer then the buffer is either full or empty.

There is a buffer empty bit in the four word table that is set if the buffer is empty. This serves to remove the ambiguity and allows a test for emptiness by referencing one word.

LBRB is equivalent to a NOP (execute the next instruction) if the empty bit is set. Otherwise it loads a byte and skips the next instruction. To load a byte the read pointer is compared to the end buffer pointer, and if they are equal the read pointer is replaced with the beginning buffer pointer. The read pointer is then incremented and the byte pointed to is loaded into the right half of the J register (the left half being cleared). If the read pointer and write pointer are equal, the buffer empty bit is set.

SBRB (30): Store Byte in Ring Buffer
If the buffer is full the instruction is a NOP, otherwise the right byte of the J register is stored, the write pointer is incremented, and the next instruction is skipped.

The I register points to a ring buffer descriptor. The write pointer is compared to the end buffer pointer, and, if they are equal, tentatively replaced with the beginning

buffer pointer, and compared with the read pointer. If the result of the comparison is equality and the buffer empty bit is not set, the pointer is not stored and the next instruction is executed. Otherwise the new pointer is incremented and stored, the buffer empty bit is reset and the right half of the J register is stored in the byte now being pointed to. The next instruction is skipped.

POT (31): Output

The device specified by the J register is selected. The word in the I register is output to this device.

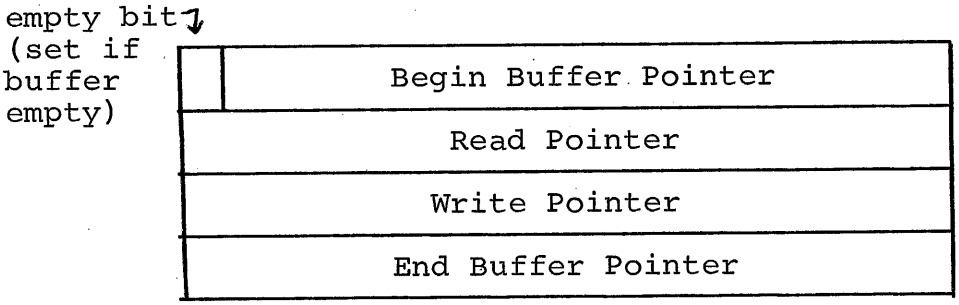
PIN (32): Input

The input from the device selected by the J register is put in the accumulator.

APPENDIX II

DATA FORMATS

Ring Buffer Pointer



Local Device Table Entries

TYPE	Char. being output
	Char. being input
Next output character (OCB)	Last input character
LVB	Flag bits

Low Speed Devices

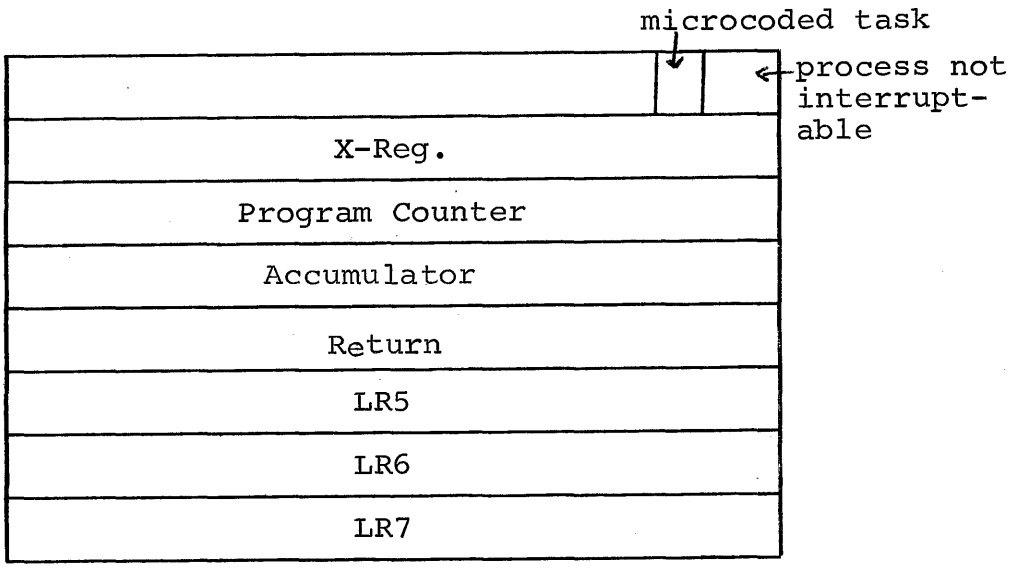
TYPE Ø	
Ring buffer descriptor pointer	
Process number	
LVB	Flag bits

High Speed Devices

Flag Bits:

- IHC: ICB has a character
- OHC: OCB has a character
- IHCC: ICB has a control character
- LEM: Local echo mode
- BS: Break strategy (2 bits)
- EBC: Echo break character
- VDT: Unknown device type

Task State Table (indexed by task number)*8



Waiting task mask (WTM)

bit 1 . . .	bit 16
bit 17 . . .	bit 32

f the N^{th} bit is on the N^{th} task is running or is waiting to run.

The local device bit table and the character type table are the same as the Phase 1 tables.