| bcc | title | prefix/class-number.revision |
|---|---|---|
| | CHIO IMPLEMENTATION PHASE 1 | IHTWD/W- 38 |

| checked | authors | approval date | revision date |
|---|---|---|---|
| checked | Paul Heckel | 12/16/69 | |
| approved | | classification: Working Paper | |
| | | distribution: Company Private | pages: 22 |

***ABSTRACT and CONTENTS***

This documents the implementation of the Phase 1 CHIO in enough detail so that someone should be able to follow the microcode. The reader of this document is expected to be familiar with the CHIO/CPU INTERFACE (FOO/S-2.1) document. In particular the appendix of that document contains the format of all of the tables used in the CHIO. If more detailed understanding is needed, this should be read along with the comments at the beginning of each sub-routine.

This document is not expected to answer all implementation questions, but it should guide the reader in finding answers.

OVERVIEW

The CHIO (Character Input/Output) Multiplexor does all of the teletype Input/Output for the system as well as character Input/Output to other devices (currently just the 360).

As far as the CPU is concerned there are some lines which characters may be written into or read from. These lines come in pairs, the first (even) line of the pair is the input line and the second (odd) line of the pair is the output line. A CPU program may read from or write into either of these lines, although it normally reads from the input line and writes into the output line.

These lines are connected to actual devices: either teletypes or 2400 baud lines (in Phase II the devices will include teletypes on the remote concentrators).

Any CPU request to these lines may be ignorant of the device type although the CPU actually sets up the device type at initialization time. The CHIO sends and receives characters to/from devices at the appropriate speed, buffers characters for both input and output, and awakens the appropriate process if the character count or the buffer count for the line underflows or overflows. The appropriate process is also awakened when a wakeup character is input.

The CHIO also echoes characters to devices in such a way as to ensure that the listing that appears on the paper is

in the logical order in which the conversation occurred rather than the actual order. I.E., if a user types ahead, a character is not actually echoed until the CPU reads it.

The CHIO also handles linking and will wakeup CCP type programs.

The primary function of the CHIO is to buffer the characters when they are input to the system until they are read by the CPU, and to buffer them from when they are output by the CPU until they are output to the teletypes.

The CHIO is divided into three parts; several low level subroutines that perform basic functions, subroutines that handle CPU requests and the logic for interfacing with the actual devices.

Examples of the first type are subroutines that read and write characters, determine the character type, or wakeup a process. Subroutines that perform CPU requests are in the second category and they call on functions in the first category. The subroutines that do bit scanning or decide whether to wakeup a process for input or output are in the last category. These also call on functions in the first category. There is no direct communication between functions in the second and third category.

CHARACTER POINTERS AND CHARACTER BUFFERS

All character buffers (including the one in the CPUIT) begin on locations divisible by 8 and are 8 words long. The first word of the buffer points to the next buffer (Ø means no next buffer). The next 7 words of the buffer contain characters.

The character buffers (with the exception of the one in the CPUIT) fall in a block of storage no larger than 64K pointed to by the scratchpad register BUFØ. The first 8 words following the location pointed to by BUFØ must contain Ø. These words may not be used as a buffer. All buffer pointers and character pointers that are stored in core are taken relative to BUFØ. Thus the CHIO must add in the contents of BUFØ to determine the actual buffer location. These relocated values are stored in the holding registers: RCP (read character pointer), RCPEND (pointer to last character to read) WCP (write character pointer.)

Character pointers are actually word pointers. All words pointed to by character pointers must be padded by leading or trailing nulls (zeros). (Thus there is a very basic reason that the null character cannot be easily changed.) When a character pointer is changed, its sign bit is set. A copy of the word pointed to by WCP (RCP) is kept in the scratchpad register WWORD (RWORD).

Given these facts the following observations can be made:

1) When a character pointer is loaded into RCP or WCP then RWORD or WWORD must be loaded.
2) When a character is read it must be deleted from RWORD (replaced with NULL).
3) If there is no character in RWORD (RWORD = Ø) then RCP must be incremented, its sign bit must be set and the new word pointed to must be put into RWORD.
4) When a pointer is stored, RWORD or WWORD must be stored but the pointer does not have to be stored unless its sign bit is set.
5) When a character is written it is placed in the right side of WWORD, or if it won't fit WWORD is stored, WCP is incremented and WWORD is set to the new character. Note that if the first character in WWORD is a NULL then WWORD (the left two characters being NULL) may be cycled left 8 to make room for a new character.
6) Characters should be stored in words right justified. If this is not done some inefficiency in the use of time or space will be the only effect.

The following facts are also true:

1) A pointer to zero is interpreted as meaning there are no more buffers to point to. Thus if this appears in the character pointer table then the line is empty.

2) The end of a character string is reached if RCP = RCPEND.

3) The count in RCNT (WCNT) is incremented for each character read (written), (nulls are not counted). This fact is used by routines like WST and RCND to determine when the correct number of characters has been read.

4) IRCH and IWCH are used to initialize read and write pointers prior to reading characters. Similarly CWCH and CRCH are called after the characters have been read.

5) RCH and WCH will read or write one character. They assume the initialization has been done and the cleanup will be done.

6) RCH1 and WCH1 will read or write one character from a line. These routines will call the initialization and the cleanup subroutines.

CHARACTER BUFFER CONVENTIONS

At initialization time the buffers must be linked together with the last buffer pointing to Ø and the pointer to the first free buffer put in the scratchpad register FREEL. NFB in the CPUIT must be initialized to the correct number of free buffers. The CHIO will keep it updated.

The CHIO always keeps a free buffer in AVB and a -1 in PRB. If at any time a buffer is needed it is taken from AVB and BFLAG is set to -1. If at any time a buffer is freed it is placed in PRB and BFLAG is set to -1. No request (with the exception of ECHO which calls BLAKE explicitly) can use or free more than one buffer. Whenever a request is finished the function BLAKE is called if BFLAG is negative to fixup the buffer situation.

RCH and WCH know to link to the next buffer if the character pointer, when incremented, is divisible by 8. They do this through subroutines stored in RBUF and WBUF. The subroutines in these registers are LNKB which links to the next buffer, freeing the old one; GETB which gets the next buffer; and ABORT (when the buffer is the CPUIT buffer and the buffer boundary should not be crossed).

BLAKE expects the number of free buffers to stay between MINFB and MAXFB (scratchpad registers) if it does not, then a CPU process called BRECHT is called to take whatever action it feels necessary.

The number of buffers that a line is allowed to obtain for output is:

XBCNT * MP / 64

where XBCNT is a quantity stored in the line and MP is a multiplier from 1 to 64. If buffer space is tight then MP should be reduced by the process BRECHT.

It is important to note that the number of buffers a process is allowed to collect is infinite, since all requests will be fulfilled. The CPU is, however, expected to stop WSTing characters to a line if NSR is set to 1 (or NIQ or NOQ is on). This means that characters will never be dropped by the CHIO.

Whenever an input (output) line acquires a buffer in excess of the number allowed, then XIB (XOB) is incremented by 1. It will be decremented by one when the buffer is returned. Because it is possible for an input line to fill up without its process servicing it, BRECHT should check to make sure that XIB and XOB have reasonable values.

The first 8 bits of the first word in the character buffer contains the line number mod 256 so that from time to time the CHIO can check the compare instruction.

When characters are to be written into a line, GWP is called, and similarly when they are to be read from a line then GRP is called. These subroutines set up WCP (or RCP

and RCPEND) and leave an interesting variable in WTCE2 (or RTCE2) for PWP (PRP). When the reads or writes to that line are finished, PWP or PRP must be called. These subroutines will update the write (read) pointers, the character counts, and check to determine whether the character count or buffer count is out of range. These subroutines may also detect abnormal conditions which cause an abort.

CPU CALLS

Someone familiar with the character and buffer manipulation functions should not have any great difficulty with the CPU calls. However, the following facts are important. CPURQ is called when there is a CPU request. It calls GLE to set up the line number and then dispatches on the type. GLE must be called to "open" a line; it will set up several interesting global variables.

When a CPU request is completed it may either return (to CPUFIN) or branch to CPUFIN. At that time M will contain the value (VALU) and NSRFLAG (which CPURQ initialized to Ø) will contain NSR.

The only functions not discussed so far that any of these functions might call are WKCQ, WAKEUP and ECHO.

WKCQ is used to determine the character type. It may be a wakeup character, a QUIT, an ESCAPE, a non-echoable character, or an ordinary everyday character for which nothing special should be done. The character type is indicated by the value ( in Z). (If WAKEUP is immediately called it will have the wakeup type setup correctly in the Z register). WKCQ does two things of some subtlety. First, it sets CHISC negative if the last character was an echoable character so that ECHO will not echo it, and it uses LCSW1 as a flag in case WKCQ is called with a SHIFT1 so that the

next time it is called it will only consider characters in the range of Ø to 37.

Note that this second condition requires that if WKCQ is called with a SHIFT1 that it must be called again before the current request is finished.

WAKEUP can be called with a variety of arguments. WAKEUP will try and get out of waking up the process if it can (WIC/WOC might not be set). If it is unable to avoid the work then it will awaken the process for the line (except for the more general entry that specifies the process to be awakened) and reset WIC or WOC to indicate this.

An understanding of ECHO requires an understanding of more of the CHIO than any other subroutine. It is responsible for echoing the current character and outputing it to any linked line. ECHO should be called if the character should be echoed assuming the echo strategy is on. Thus the calling subroutine must determine whether the character should be echoed now, has been echoed or should be echoed in the future. ECHO first saves WCNT, WCP, WWORD, LINE and BFLAG. BFLAG is saved so that the buffer count in the line will be correct; the other words are saved so that writing of characters may proceed as if ECHO was not called. The read character variables do not have to be saved because ECHO does not call any character reading functions.

If the echo strategy to the line is on, and the line is an output line, then ECHO calls GLE with the output line number (one plus the current line number) and calls WCH1 to write the character in the line. If the LCWS1 (last character was a SHIFT1) flag is on, the character is echoed as SHIFT1 followed by the character plus 40B. If the character is a control character (which could only be a SHIFT1) then the character is not echoed.

ECHO then looks to see if the link bit is on, and if it is it will then write the character into the linked line. The saved variables are now restored and GLE is called to reinitialize those variables associated with the line. Intermixed in this code are calls to BLAKE any place a buffer might be freed or used.

HIGH SPEED LINES

If there is a request from a high speed line then B2400 is called. This subroutine will get an input character and call INLOGIC to put it in the appropriate line, or call OUTLOGIC to get an output character from a CPU line to output to the 2400 Baud line.

GMTL is a subroutine that determines local line number and CPU line number and sets up the necessary global variables.

INLOGIC is a subroutine that writes an input character on an input line. It is on the same logical level as say WST or RSTB in that it makes all of the decisions but does none of the work. It will echo characters and wakeup the process for the line if necessary. Similarly OUTLOGIC gets an output character from the current line. Neither of these subroutines calls any subroutine not yet described.

OUTL2 and INL2 are similar to OUTLOGIC and INLOGIC except they are used when a non-control character (possibly in the range 0 to 37B) is desired. OUTLOGIC might return a SHIFT1 but OUTL2 will call OUTLOGIC again to determine the absolute value of the shifted character. A is set if the character that OUTL2 returns is a control character (which will not be SHIFT1). INL2, if it has a character from Ø to 37 as an argument, will call INLOGIC twice, the first time with a SHIFT1 and the second time with a 40B plus the character as the argument.

CONTROL CHARACTERS

HOW TO FIT 32 CHIO CONTROL CHARACTERS AND 256 CHARACTERS IN 8 BITS.

The first 32 characters are reserved for CHIO control characters. That means that the CPU is not allowed to send a character in the range of Ø to 37B to the CHIO without adding 40B to it and preceding it by a SHIFT1 (unless it means it as a control character). This unfortunately makes the CHIO's life harder than it might otherwise be, and some subtle effects occur.

First it should be noted that RSTB and RCND must assume that if they ask for N characters and the Nth character is a SHIFT1 that N + 1 characters will be delivered. This is necessary because otherwise the CHIO would not know whether to echo the SHIFT1 until the next character was read. It is also rather unesthetic to deliver half a character to the CPU.

The CHIO has a few routines that try to make life simpler for other routines by hiding the control character problem as much as possible. OUTL2 and INL2 are two examples. Unfortunately there are some subroutines that have to worry about the problem. The most important of these are WKCQ, ECHO, INLOGIC, OUTLOGIC, INL2, OUTL2, RSTB and RCND. The most basic subroutines like RCH and WCH are not aware of anything but 8 bit characters.

BIT SCANNING

Each 1/7 of a character time the subroutine BSCN is called. BSCN really has three incarnations and should be considered a completely separate subroutine for each device type: Model 35, Model 37 and IBM 2741. Considering Model 35s, for example, when BSCN is called it has three separate tasks to consider: it must look for an incoming first bit on idle Model 35 teletypes, it must input the next bit on a subset of input teletypes, and it must output the next bit on a subset of output teletypes for Model 35s. The local device bit table is divided into three sections, one for each device type. The section for Model 35s is selected. An entry, BSNO, which contains the bit slice number MOD 7 is incremented MOD 7 each time BSCN is called for Model 35s (every 1/7 of a Model 35 bit time, about 1/7*11Ø) of a second). BSNO can therefore be used to determine which 1/7 of a bit slice is being processed.

In the LOCAL DEVICE BIT TABLE there is an entry, NCIP, which has a bit on for each teletype that is a Model 35, and that has no input in process. This word is masked (obtained by a PIN) with the word of bits for the teletypes with bits on. If any previously idle, (for input) teletype has had a bit turned on then the resulting word is non-zero. In this case FLB is called. It will find out what the teletype number involved is and set up some interesting variables. IFB is then called to modify the Local Device Bit Table so

that 3/7 of a bit time later DID will find the bit for that teletype on. It does this by adding (mod 7) 3 to BSNO and setting the bit for the selected teletype in the BSNOth input word.

LDVTE has 7 word pairs for each device. The first word in each pair is for the input teletype lines, and the second is for the output teletype lines. The pair-word is selected by the value of BSNO. The Nth bit in pair-word for input means the teletype input bit should be sampled. Similarly the Nth bit for output means the next bit for the Nth teletype should be output. DID is called for the input case, and DOD for the output case. DID keeps collecting bits in the Local Device Table until it has a whole character. Then it calls INL2 to put it in the input line. DOD behaves analogously outputing bits until there are no more to be output and then calling OUTL2 to get the next character.

An important observation is that any request that involves a character in the range of Ø to 37B must eventually call RCH twice, once for the SHIFT1 and once with the character + 40B to ensure that WKCQ will not be left hanging. This is meant as a warning; the two places where bugs of the "you touch something here and something else pops up over there" variety are likely to occur with SHIFT1 and with ECHO.

CLOCK DRIVEN FUNCTIONS

The top level routine (GNR; Get Next Request) recognizes three types of requests; CPU requests, 2400 baud line requests, and clock driven requests.

There is a list of scratchpad registers from NCI to LCI. Each of these registers has a subroutine number in the upper 5 bits and a time in the lower 19 bits. These registers are kept sorted by the time (in the lower 19 bits) and when the current time is equal to or greater than the time in NCI then the subroutine specified by the high order 5 bits of NCI is called. This subroutine has the responsibility of resetting NCI (by incrementing it) to the next time that it (the subroutine) should be called so that GNRFIN can sort the registers on return.

Two subroutines are of particular interest: SUBNUL which does nothing but make sure that it will not be called again soon, (this can be used to turn off a device type quite easily) and CKFXP which is called whenever all of the sub-routines have the high order time bit set. This subroutine turns off all of these high order time bits so that the strict ordering of the scratchpad registers will always hold.

The only other subroutines that are called are RTUPDATE which updates the Reat Time (RTLOW, RTHIGH) in core each milli-second and PRUPDATE which resets a PLINE for GPR each 1/2 seconds.

THE CHIO'S INTERFACE WITH THE REST OF THE WORLD

The CHIO receives a STROBE1 from the CPU when it gives the CHIO request, and a STROBE2 from the UTP when the UTP wants the CHIO to prepare to crash. If the UTP (or someone else) sends a Z.M. signal to the CHIO, it goes into system restart mode (unless BREAKPOINT is on). The CHIO will wait until it receives a STROBE1 from the UTP to load its scratchpad and go to the GNR. Note that the meaning of a STROBE1 is context dependent because the CHIO cannot tell where it came from.

When the CHIO wakes up a process (as specified in VSI/W-14) or when the CHIO updates the real time clock in core (once per millisecond) it sends a STROBE1 to the UTP. The core locations read or written by the CHIO are 40B to 77B and the Wakeup Table of the UTP.

THE MECHANICS OF MODIFYING AND COMPILING THE CHIO

The CHIO consists of two files; DEFS, and CSYS. DEFS, which contains the CHIO's definitions, may be used by three different programs.

First, if it is used to make a DUMP file of MICRO, it will contain all of the definitions needed to compile CSYS. This includes field definitions, constants, scratchpad register assignments, and holding register assignments, and branch and special condition definitions.

Second, if used as a QSPL include file (following QINIT as in QPGM) it will define all the above as QSPL constants except for the branch and special conditions. The scratchpad registers are given their number as their value, thus MTAA is 1, for example. The fields defined may be used in the resulting QSPL program. Two macros of interest are defined: SPLF and SPLFC. The first will, if given a field name as an argument, have as value a SPL field descriptor that points to the CPU line table and can be used in a call to GETFIELD or PUTFIELD. The second produces a field descriptor that points to the character pointer table.

Thirdly, DEFS may be used as an input file to the SNOBOL program TABLE to produce the appendix to the CHIO/CPU Interface document (FREE DOCUMENTATION!)

CCSYS is a CCP-type SNOBOL program that will compile the

communications system. If you want to know what is happening just after

START patch in;

NORMALSW = 'ON'

SWITCH = 'ON'

and you will be able to overhear the conversation that it has with the CPU.

A BRIEF DESCRIPTION OF EACH OF THE TABLES IN THE SYSTEM

The CPU Line Table is a table consisting of a four word entry for each input/output line pair. It contains all of the information pertinent to both the input and the output line.

The Character Pointer Table consists of a three word entry for each line, the input line followed by the output line. This table contains character and buffer counts and the character pointers.

The Device Table contains the escape character for each device and the character type for each of the 256 characters. The character TYPE (see document AKOCS/M-15) is a number from Ø to 3 that indicates whether the character is an alphanumeric, a punctuator, an echoable control character like carriage return, or a non-echoable character.

The CPU Interface Table is the name given to the block of core that the CPU and the CHIO use to communicate with.

The Local Device Buffer Table is used when servicing requests from the devices rather than requests from the CPU. It contains the CPU line number so that the line can be "opened". The teletype bit scanning subroutines also use it to keep characters that are being bit scanned.

The Local Device Bit Table is explained in the section on bit scanning.