bcc	title	prefix/class-number.revision
	INTERRUPT AND WAKE-UP SYSTEM	IWS/W-11.1

checked	authors	approval date	revision date
Billie W Hampton	Rainer Schulz		11/6/69
checked: Larry L Barnes		classification: Working Document	
approved: Mel		distribution: Company Private	pages: 13

ABSTRACT and CONTENTS

This document describes the interrupt and wake-up system. It also covers non-interruptability of sub-processes and gives a description of the calls concerning the above mentioned system functions.

Interrupts and Wake-Up

Introduction

The M1 system provides three mechanisms for inter-process communication:

1) shared memory pages, which must reside in files
2) CHIO lines
3) the interrupt/wakeup system (IWS) described in this document

The intention behind the system design is that files will be used for passing data between processes and that interrupts will serve to start the execution of a process or to forcibly divert its attention. No mechanism is provided for passing data with an interrupt or wakeup. It is not intended that CHIO lines should normally be used for inter-process communication.

The IWS also allows interrupts to be generated as a result of the elapse of real time or compute time.

Interrupt Cells

Each process has a number of interrupt cells. These are allocated and controlled exactly like PMT bytes. Each one thus contains a control lock. It also contains other information which is shown in FIG. 1.

Each cell corresponds to a bit in the process interrupt word (PIW). (See below.) The action field (as shown in FIG. 1) tells what to do if the interrupt becomes effective. The following choices are offered:

do nothing

call subprocess n

generate trap n

The source tells where the interrupt comes from. The following choices are offered:

from setting the bit in PIW which corresponds to the cell

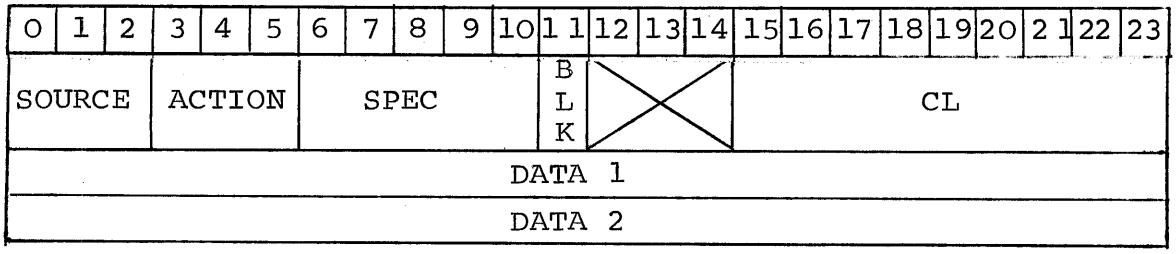
from arrival of real time (rt), measured in 1 ms units

from elapse of ct ms of billable time

There are 23 interrupt cells.

The first nine bits in PIW (nine interrupt cells) have been reserved for special interrupts which are described in the next section. The other cells can be set by the user to timer interrupts or merely to the occurrence of the corresponding interrupt bit in PIW.

INTERRUPT CHANNEL TABLE (ICT)



Let i be index into ICT

SOURCE	ØØØ	None
	ØØ1	Occurrence of Interrupt i
	Ø1Ø	Real Time Interrupt
	Ø11	Compute Time Interrupt
	1ØØ	Unused
	1Ø1	
	11Ø	
	111	

ACTION	ØØØ	No action - ignore interrupt
	ØØ1	Call specified SP
	Ø1Ø	Generate specified trap
	Ø11	Unused
	1ØØ	
	1Ø1	
	11Ø	
	111	

SPEC - SPT index if ACTION = ØØ1

TRAP number if ACTION = Ø1Ø

DATA Clock value if COND = Ø1Ø or Ø11

BLK BLOCKING ON THIS BIT IS ALLOWED

CL CONTROL LOCK

FIG. 1

The interrupt becomes effective when the condition specified by the source appears, unless the process is non-interruptable; for the handling of this situation, see below. When it becomes effective, the specified action is taken, and the interrupt is then forgotten. If it was caused by a PIW bit, the bit is cleared unless the action was 'do nothing.'

Note that each interrupt cell is associated with a particular PIW bit; it may be used to respond to the setting of that bit, or for a real or a compute timer. There is no restriction on the number of interrupt cells used for timers.

The Process Interrupt Word

Another object which is associated with a process in order to implement the IWS is a single word called the PIW. The setting of some of the bits in the word (called settable bits) may cause an interrupt as described above. They can be set in the following ways:

If a process is open, any settable bit can be set.

The bits not marked with a * in the description of the PIW below can be set by the conditions listed in the description. They are also settable by other processes and can be used as wake-up conditions.

The ES bit can only be set by another process if the sub-process making the call to set the bit controls the ES bit in its own process.

The bits marked with a * are not settable. They cause special actions to be taken as described below. They cannot be used for wake-up conditions.

PIW:

			CO	ES	QT	AMC	CHI			General Settable Bits	RAP	RSI	RT
Ø	1	2	3	4	5	6	7	8	9	1Ø			23

*RT real time elapsed. The hardware maintains one real timer per process, which is then multiplexed by the process. This bit is used by the system to signal the occurrence of the specified time; it is interpreted by the monitor's machinery for doing the multiplexing and is irrelevant to any user program

Bits 9-22 INTERRUPTS GENERATED BY OTHER PROCESSES AND TIMERS

*AMC interrupt generated by the AMC.

CHI interrupt generated by the CHIO.

QT quit character received; generated by the CHIO.

ES escape character received; generated by the CHIO.

*CO carrier off received; generated by CHIO. Cannot be set by a process, but is otherwise treated like any settable bit.

Bits 0-2 These bits are reserved for future fixed interrupts.

The bit positions of the above mentioned interrupts are subject to change without notice. A call to the system is provided to convert a character constant (e.g., "AMC") to the proper bit position in the interrupt word. The abbreviations shown above reflect the proper character constants for the interrupts.

Wakeups

Some bits in PIW can be used for wakeups as well as for interrupts (see above). If a bit in PIW is to be used for wakeup only, the action in the interrupt cell should be "do nothing." In particular, the monitor call which blocks a process accepts an integer argument which is used as a mask for the PIW. When the process is woken up by the system, the monitor's block routine proceeds as follows:

1) Merge the PIW in the PRT with an extended PIW kept in the context block and clear the PIW. This is done so that the system can know that the process has received the information passed to it in the resident PIW. All the operations which use the PIW actually work on the EPIW, which will therefore not be mentioned again.
2) Do the necessary interrupt processing. At this time real-time and compute-time interrupts get merged into PIW. The non-interruptability trap also gets set (see non-interruptability below).
3) If the process is interruptable and if (TEM $\leftarrow$ PIW $\wedge$ $\overline{\text{WAKE-UP MASK}}$) $\neq \emptyset$, then cause highest priority interrupt in TEM. If the action of the interrupt is "do nothing," look at next bit in TEM. If no bits in TEM cause any action, go to next step. If some bit (I) causes some action (trap, or interrupt) then

   $$\text{PIW} \leftarrow \text{PIW} \wedge \bar{I} \wedge \overline{\text{WAKE-UP MASK}}$$

4) If the non-interruptable trap is set, cause trap and reset NI bit for sub-process.

5) If PIW $\wedge$ WAKE-UP MASK $\neq \emptyset$ then

$$PIW \leftarrow PIW \wedge \overline{\text{WAKE-UP MASK}},$$

and return to user.

The effect of all this is that PIW bits can be used as wakeup-waiting flags, in addition to being used for interrupts. Once set, a PIW bit will not be cleared until it causes a wakeup or interrupt or is cleared explicitly. Wakeups take precedence over interrupts (if both interrupts and wake-ups can occur) because of the processing described above. If a process is blocked waiting for a bit, however, and another bit comes on and causes an interrupt, the interrupt will occur.

A sub-process cannot block on a bit in PIW if the corresponding ICT entry BLK bit is not set.

Non-interruptability

It is necessary to have some mechanism by which a process can prevent itself from being interrupted, so that processing which must take place without interruption can occur. To this end a process may declare itself non-interruptable, giving two parameters: the amount of compute time and the number of disk accesses before it becomes interruptable again. These numbers are converted into a real-time figure. The minimum of this figure and a fixed limit (say 2 minutes) is added to the current real time to obtain the end E of the non-interruptable period.

During this period interrupts which occur do not become effective, but are remembered. The mechanism for this is simple: the PIW bits for interrupts are not reset. When the process becomes interruptable again, the interrupts will occur in the order that they appear in PIW, where the most significant bit has the highest priority.

The limit E is actually associated with a call stack entry. When a call is made, E is copied to the new entry. The called subprocess can extend it once. When a return occurs, it reverts to its former value. If the limit is reached, a NILE trap occurs and the process becomes interruptable; this happens whether or not an interrupt is pending. The NILE trap of course indicates an error. The normal way to become interruptable is to make a system call with that effect; if S makes this call and S was called by T and T was non-interruptable

with limit E_T, then Es becomes E_T and the process remains non-interruptable. The purpose of all this machinery is to allow sub-processes to set NI without worrying about whether their caller's have set it, while still preventing the process from becoming permanently lost. The limits are made large because they are expected to be needed only during debugging; a user whose program does not explicitly set NI should never experience such delays, since the (debugged) programs he calls will presumably reset it expeditiously.

For obvious reasons the subprocess called as a result of an interrupt is made NI with the maximum limit.

Operations on ICT

The numbers in front of the function descriptions correspond to actual MCALL numbers.

165) Convert interrupt character constant to interrupt number, CVINT(CH).

This function expects the character constant CH to be one of the defined bits in PIW. The function returns the PIW bit position of the interrupt.

The call fails:

a) if CH is not defined in the system.

166) Read interrupt cell, RDICT(N, ARRAY WA)

This call is always legal. It reads interrupt cell N and returns its value which consists of 3 words.

The call fails unless:

a) -1<N<12

167) Read PIW, RDPIW()

This call is always legal

168) Make interruptable, RESNI()

Makes process interruptable, unless there are some entries in the call stack whose NI has not expired yet.

This call is always legal.

169) Make non-interruptable, MKNI(CT,DA)

Makes the process non-interruptable for a real time derived from compute time CT and number of disk accesses DA, or for 2 minutes, whichever is less.

The call fails if:

a) NI is already set

170) Block (M), BLOCK(M)

Blocks the process until M∧PIW≠Ø; then sets PIW ← $\overline{M}$∧PIW.

The call fails if:

a) Any interrupt cells corresponding to bits in M do not have the BLK bit set or do not contain an entry.

XXX) Set PIW bits (P,M), SETPIW(P,M)

If process S controls process P, it can set any bits in PIW of process P. P is an index in OFT. This means that the process has to be opened before this call is made.

If process S = process P, then the bits in M get merged into PIW if the calling sub-process controls all corresponding interrupt cells.

171) Clear PIW bits (M), CLPIW(M)

This call does PIW ← $\overline{M}$∧PIW

The call fails if:

a) any of the bits in M correspond to interrupt cells which are not controlled by the calling sub-process.

172) Set interrupt cell N, SETICT(N, SAB, SP, LONG DA)

The source, action and BLK bit are taken out of SAB. They have to be in the positions specified for the ICT table. The SPEC in ICT is set to SP, and the data words (if the ICT cell is a timer) are set to DA.

The call fails:

a) unless -1<N<12

b) if KEY(S) ∧ ICT[N]$CL=Ø

c) unless SOURCE<4

d) if source is timer and N<9

e) unless ACTION<4

f) unless NAME(SPEC) ∧ KEY(S)≠Ø if ACTION=1

173) Set CL in interrupt cell N, SCLICT(N,CL)

Sets the control lock in interrupt cell N.

The call fails unless:

a) -1<N<12

b) KEY(S) ∧ ICT[N]$CL≠Ø

174) Acquire interrupt cell (N). ACQICT(N)

Acquires cell N, or some available cell if N=-1.

The CL of the cell is set to NAME(S) and ACTION is set to zero (Ø).

It returns the ICT entry number on normal return.

The call fails unless:

a) -2<N<12

b) ICT[N]$CL=Ø

c) no more free entries

If N = -1, then the search for available interrupt cells starts with the low order bits.