A proposal for the values of a call stack entry and their manipulation by the return operations.

I) variables maintained in a stack entry

- p-counter (pc)
- p-wait modifier (pcm)
- F-return count (FRTN)

class code for cys process at top of stack
Top of stack class code
IP list pointer
Interrupt inhibited

This proposal only concerns the first 3.

A) The p-counter contains a non-negative integer.
B) The p-wait modifier has one of 3 values
   i) about to execute instruction at address pc
   ii) in the middle of a complicated instruction at address pc
   iii) almost finished with a complicated instruction at address pc
C) The F-return count contains a non-negative integer
II) Variables maintained in a set process descriptor:

- BIT 1: interrupts armed
- BIT 2: interrupts inhibited
- BIT 3: interrupt waiting
- BIT 4: interrupt allow
- BIT 5: error selection mask

Instead
III) Fancy return instruction

a) parameters as follows:

\begin{align*}
PL & 0 : \text{key} \\
P2 & 80 : \text{data parameter being returned, if any} \\
P3 & 8c : \text{capability parameter being returned, if any} \\
P4 & 0 : \text{error return, if any} \\
P5 & 0 : \text{error number, if any} \\
P6 & 0 : \text{interrupt return, if any} \\
\end{align*}

b) the return action tests various bits in the key and performs actions as described below

1) interrupt bit

2) if processor has interrupt waiting (not inhibited?)

3) remove the interrupt (not represented symbol to separate?)

4) interrupt b.t.

5) if, after a scan is started down the tree towards the root of the beginning of the subprocess at top of stack, looking for a subprocess

\[ ... ? \]
with interrupts armed, when one is hand, this interrupt
making if its interrupt waiting bit is on, no further action
for the interrupt, If off, its interrupt waiting
bit is turned on and PC is placed in the subprocess interrupt
datum.

ii) return with previous bit.

if on, the environment is restored to that of Fparent
stack, and appropriate parameter return action
takes place, including making a new FOS in

iii) finish bit.

if on, the Finish count is incremented by 1.

iv) repeat bit.

if on, The pointer modifier is set to, "about to execute
next instruction at address pc".

v) complete bit.

if on, the pointer modifier is set to, "almost finished with
complete instruction at address pc". (note that this
bit overrides the repeat bit.)
VI) \textit{Error Bit}

If only a scan is started down the tree towards the root, beginning with the subprocess at top of stack, looking for a subprocess with the bit on in its error selection mask corresponding to the given error class, when one is found, the bit is turned off in its error selection mask and a new top of stack entry is made, pushing down the old top of stack entry. The new top of stack entry represents a call on the subprocess just located, i.e., its \texttt{p-counter} is set to the entry point of the just found subprocess, \texttt{pcr} is set to “about to execute an instruction at address \texttt{pc}, its \texttt{f-return count} is set to 0, its class code is set to that of the found subprocess, its top of path class code is set in the usual manner, the environment is now set to that of the new top of stack and the error class and number are placed in the appropriate locations.

\textit{Note:} that the error bit is just one scan value, since it may create a new stack entry. This could be avoided if the error into would be held in the subprocess descriptor as for interrupts, or held in the stack entry itself.
(c) steering next, the return operation begins a scan towards the root.

(c) Now the return action starts a scan down the tree towards the root beginning with

(i) "about to execute an instruction at address pc" a scan is started towards the root of the tree.

Towards the root beginning with the top of the stack the subprocess looking for a subprocess with interrupt waiting bit on. If one is found and the subprocess is not the top of stack subprocess or if it is the top of stack subprocess and its interrupt inhibit bit is off then the following takes place:

Interrupt waiting bit is turned off. Interrupt inhibited is turned on. A new top of stack entry is made as in the error bit case. The environment is set to that of the new top of stack. The interrupt datum is copied from the subprocess descriptor to the appropriate place in core.

Finally we go to c) (hence c) i) EPC = interrupt - $s$
ii) "in middle of a complicated instruction at address pc" 
  
  Set the environment to the top of stack. From the XR instruction locate the appropriate operation. 
  Now set the operation as sufficient depth to handle the F-return continuation (0 is for store, etc.) 
  If so, make a new stack entry in environment, 
  (Set pc to entry point, set c (hence c i i)) 
  If not, set pc = pc + 1, set pm to "about to execute an instruction at address pc" 
  (hence c i i)

iii) "all finished with a complicated instruction at address pc" 
  
  osystem return 
  
  Set the environment to the top of stack. From the XR instruction compute the appropriate pointer offset. 
  If this is within range set pc to the new value, set pm to "about to execute an instruction at pc" and set c (hence c i i) 
  If this is not within range, generate an appropriate error, and an error can occur in the execution case. Then set c (again, hence c i i)
our existing return instructions act same as his one bit w. a subset of the specified params and in a fixed way. (could fill the unused params?)

4) ordinary return
   only bit on is complete bit. No params

5) F-return
   only bit on is F-return. No params

6) error-return
   only bit is error-return. Error class and error number are only params

7) return with params.
   complete bit and params bits are only bit on.
   Block data and block capability are only params.

8) Special return
   repeat bit only bit on. No params

) used on a call, want to make a new stack entry, set pc = curnt reg
pcm = "about to execute an instruction at pc" new procedures
in II (III) C) (D) (hence III) (C) (II)
A) an external interrupt arises with a datum and a class code.

3 purposes:

1) The given sub process is examined if interrupt unit is on, nothing else the datum is stored in the sub process and interrupt unit is turned on.

2) Two sono sub process is examined. If interrupt not normal, then nothing, else as in 1)

3) After a timeout occurs (the root is started with the given sub process. If a sub process is found with interrupt armed then proceed as in 1), otherwise?

B) Now examine ke process

1) Assume event channel just not received an event, change pc in top stack entry to "about to execute an instruction at pc", set status in the process descriptor to "not ready from event channel", reschedule the process.
ii) Hung on event channels has received an event.

Could have been done by an event itself? 

should be?

iii) Swapped out. Is quantum overflow?

denoting

c) one further thing

When every process is swapping or after removal from any event channels, a timescan towards the top of stack. Subprocess is done, looking for a subprocess with interrupt unit. It has interrupt inhibit effect if it is not top of stack. If so, it is called as in (c) of (iii).
a number of unclear things

6) stain "too full"

b) what if 2 subprocesses found with interrupt waiting? one has interrupts inhibited, etc.

hmm. If 1st one sets called, as command then ensued, 2nd would find 2nd, so if 1st inhibited

\[\text{I think the run should continue and find 2nd.}\]

\[\text{Also, this would happen if the hypoglycemia did not have interrupt setting enabled.}\]

\[\text{What if no subprocess found with interrupt enabled?}\]

c) no subprocess found with error class 6? then in it's work?

d) no subprocess found with error class 6? then in it's work?

\[\text{need}\]

\[\text{Set interrupt inhibit}\]
\[\text{Clear interrupt inhibit}\]
\[\text{Set interrupt armed}\]
\[\text{Clear interrupt armed}\]

\[\text{This description can probably be somewhat better reformulated especially if we make new stack entry items}\]