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

1) variables maintained in a stack entry
   - p-counter (PC)
   - p-counter modifier (PCM)
   - F-return count (FRN)
   - class area for suspensions at top of stack
   - Top of stack class code

This proposal only concerns the first 3.

A) The p-counter contains a non-negative integer
B) The p-counter modifier has one of 3 values
   i) want to execute instruction at address PC
   ii) in the middle of a complicated x's instruction at address PC
   iii) almost finished with a complicated x's instruction at address PC
C) The F-return count contains a non-negative integer
II) Variables maintained in a sub process descriptor

BIT: Interrupt armed
BIT: Interrupt inhibited
BIT: Interrupt waiting
D: Interrupt duty
D: Error election mask
III) Fancy return instruction

1) parameters as follows:

\[\begin{align*}
\text{P10} & \quad \text{E9} & : & \quad \text{key} \\
\text{P2} & \quad \text{BO} & : & \quad \text{data parameters being returned, if any} \\
\text{P3} & \quad \text{BC} & : & \quad \text{capability parameters being returned, if any} \\
\text{P4} & \quad \text{O} & : & \quad \text{error cues, if any} \\
\text{P5} & \quad \text{O} & : & \quad \text{error number, if any} \\
\text{P6} & \quad \text{O} & : & \quad \text{interrupt return, if any} \\
\end{align*}\]

2) The return action tests various bits in the key and performs actions as described below.

\[\text{Interrupt bit} \quad \text{if any}\]

3) The return action tests various bits in the key and performs actions as described below. In what follows, the top of stack entry is the one immediately below the stack entry for the subprocess executing the return operation.

\[\text{Interrupt bit} \quad \text{if any}\]

Thus a scan is started down the tree towards the root from the beginning of the subprocess at top of stack, looking for a subprocess...
with interrupts armed, when one is found, its interrupt
masking if its interrupt waiting bit is on, no further action
for the interrupt. If off, its interrupt waiting
bit is turned on and processed. The interrupt
datum is placed in the suspensions interrupt
datum.

ii) return w/ parent w/ b.

if on, the environment is returned to out of stack
store, and appropriate parameter return action
takes place.

iii) return b.t.

if on, the return count is incremented by 1.

iv) repeat b.t.

if on, the pointer modifier is set to, "about to execute
an instruction at address PC".

v) complete b.t.

if on, the pointer modifier is set to, "almost finished with
an instruction at address PC". (Note that high
bit overrides the repeat bit.)
VI) error bit

If only a scan is started down the tree towards the root
beginning with the subprocess at top of stack, looking
for 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 p-counter
is set to the entry point of the found subprocess.
Its p-counter
is set to 0, "about to execute an instruction
at address pc." Its f-return count is set to 0,
Its class add is set to that of the found subprocess.
Its top of path class add 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.

Notice that the error bit is lost on scan and since it
may create a new stack entry. This could be avoided if
the error info could be held in the subprocess descriptor
as for interrupts or held in a stack entry itself.)
6) *Finishing Next* The return operation begins as soon *as possible*

The root

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

C) Now the return action examines the top of stack entry.
   (This is a new one if the error occurs.) It branches
   on the value of the pcm bit.

1) "about to execute an instruction at address pc" a scan is started down the tree towards the root beginning with the top of stack sub process looking for a sub process with interrupt waiting bit on. If one is found, and its sub process is not the top of stack sub process, or it is the top of stack sub process and its interrupt inhibit bit is off, then the following takes place:

   Interrupt waiting bit is turned off. Interrupt inhibit 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 data is copied from the sub process descriptor to an appropriate place in core. Finally we go to C) (hence C) 1) [pc = interrupt -8]"
ii) "in middle of a complicated xR instruction at address pc"
set the environment to the top of stack, from the xR instruction locate an appropriate operation, now see if the operation has sufficient depth to handle an E-return and instance (0 is for 1st one, etc)
if so, make new stack entry as in execute case, (set pc to entry point), go to c) (hence c)i,)
if not, set pc = pc + 1, set pm to "about to execute an instruction at address pc", go to c)
(hence c)ii)

iii) "almost finished with complicated xR instruction at address pc"
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 location, set pm to "about to execute an instruction at pc" and go to c)
(hence c)i,)
If it is not within range generate an appropriate error, and an error can only be re-executed. Thus go to c)
(again, hence c)ii)
IV) our existing return instructions act same as his one bit in the subset of the specified params and with fixed 0123 (could fill unused params?)

a) ordinary return
   only bit on is complete bit, no params

b) f-return
   only bit on is f-return, no params

c) error-return
   only bit on is error-return, error class and error number are only params

d) return with params
   complete bit and params bit's are only bit on.
   both data and both capability are only params.

e) special return
   repeat bit only bit on, no params

IV) when a call want to make new return entry, set pc = only bit
    pcm = "what to execute on instruction at pc" for procedures
    in III) C) D) (hence III) C) I) )
external interrupt.

A) an external interrupt arrives with a daemon and a process over.

3 proposals

1) The given sub process is examined. If interrupt waiting is on, nothing; else the daemon stored in the sub process and interrupt waiting is turned on.

2) The given sub process is examined. If interrupt not armed, then nothing, else as in 1)

3) do a freeze up to the next level. 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 the process

1) Any or an event channel, just yet received an event, change pc in top stack entry to “about to execute an instruction at pc”, set signal in the process descriptor to unknown from the event channel, reschedule the process.
1) Hung an event channel, has received an event.
Could have been done by the event itself?

ii) shuttered? Could quantum overflow?

C) one for now thing

When every process is suppose to be removed from any event channel, a time scan towards the loop starting with the top of stack subprocess is done, looking for a subprocess with interrupt waiting on processor.

If one is found, a check is made to see if it was interrupted inhibit off or if it is not top of stack. If so, it is called as in C) I) of III)
a number of unclear things

b) stay "top fill"

c) what if 2 subprocess found with interrupt waiting? one was interrupts inhibited etc.

hmm. If 1st one sets call and as commercial then ensure that would find 2nd. So if 1st inhibited

I think this can should continue and find 2nd. ? Also, this would have if the top guy that did not have interrupt waiting even if he had inhibited.

c) what if no subprocess found with interrupt on addr?

d) no subprocess found with error exists? then in it's most?

VIII) need

set interrupt in 6.1 bit
Clear interrupt in 6.1 bit.

set interrupt around
Clear interrupt around

IX) (This description can probably better subroutine especially the move new stack entry items)