SECTION 14

VERIFYING A SCHEDULER

ON WHY IT IS IMPOSSIBLE TO PROVE THAT THE BDX930 DISPATCHER IMPLEMENTS A TIME-SHARING SYSTEM

Robert S. Boyer
J Strother Moore

The SIFT system, as coded by Chuck Weinstock, is all written in Pascal except for about a page of machine code. The reason that machine code is used at all is that the SIFT system implements a small time-sharing system in which Pascal programs for separate application tasks are executed according to a schedule with real-time constraints. The Pascal language has no provision for handling the notion of an "interrupt" such as the B930 clock interrupt. The Pascal language also lacks the notion of running a Pascal subroutine for a given amount of time, suspending it, saving away the suspension, and later activating the suspension. Machine code was used to overcome these inadequacies of Pascal (and most other higher order languages). Code which handles clock interrupts and suspends processes is called a dispatcher.

The BDX930 SIFT dispatcher consists of the following 14 BDX930 instructions.

```
CINT    PUSHF   15              save the flags
        PUSHM   1,13            Save registers
        PUSHM   0,0             and the resume address
        LOAD    0,ACLK          indicate a clock tick
SCHG    TRA     1,15            save the current stack pointer
        LDM     15,15,STACK     point at the "exec" stack
        PUSHM   0,1             set function code and resume stack
        JSS*    ASCHE           call the scheduler which is a pascal function
        TRA     15,12           that returns the new tasks r15 value.
        POPM    0,0             restore the resume PC to R0
        POPM    1,13            restore some registers.
        POPF    15              and the flags
        CONT    ES              allow interrupts
        RET     0               and go resume this routine
```

When the current task is interrupted by a clock interrupt before normal termination, control is transferred to CINT by the clock interrupt mechanism. The code at CINT pushes onto the task's Pascal stack the current flags, registers, and pc, and sets a flag in register 0 to indicate that the task was interrupted prematurely. Control then reaches SCHG.

On the other hand, when the current task terminates normally, code not shown here does the following: the clock interrupt mechanism is disabled, the necessary reinitialization information is saved on its Pascal stack, register 0 is set to indicate that the task was terminated normally, and control is transferred to SCHG.

In either case, the dispatcher then saves the task's stack pointer, reinstates a stack pointer used exclusively by the dispatcher and scheduler and jumps off to the Pascal code for the scheduler. The scheduler stores in the task table the task's stack and saved state information. The scheduler returns to the dispatcher the stack and state information for the next task to be run. The dispatcher then reinstates the flags, registers, and pc for that task, enables interrupts, and returns to the task.

Thus, the dispatcher and scheduler apparently implement a time-sharing system in which each user task is running on a "virtual" BDX930. We set out to try to prove that the 14 lines of code above correctly implemented those virtual BDX930s.

A reasonably simple specification of the time-sharing system goes something like this: The real BDX930 is supporting n vitual BDX930s, each devoted to a different user task. The virtual BDX930s are identical to the real BDX930 except for the absence of clock interrupts and certain parts of memory. A "mapping function" can be defined that maps the state of the real machine into an n-tuple of states of the virtual machines. When an instruction is executed on the real machine either the n-tuple of virtual states is unchanged or else one of the virtual states is advanced by one instruction and the remaining states are unchanged.

To capture the semantics of the instruction set, we encoded in our logic a recursive function that describes the state changes induced by each BDX930 instruction. Thirty pages are required to describe the top level driver and the state changes induced by each instruction (in terms of certain still undefined bit-level functions such as the 8-bit signed addition function). We encountered difficulty getting the mechanical theorem-prover to process such a large definition. However, the system was improved and the function was eventually admitted. We still anticipate great difficulty proving anything about the function because of its large size. However, the problems that have stopped us have nothing to do with mechanical proof; instead they are in formalizing a suitable specification.

We discuss three problems below: specifying the interrupt mechanism on the BDX930, specifying the mapping function, the specifying the restrictions on user tasks.

Interrupts: Clock interrupts on the BDX930 occur at a specified interval. But it is difficult to get precise statements regarding how long any given instruction will take. The situation is complicated by the fact that some cycles are stolen to service writes to the data file by concurrent processors, thus introducing a true nondeterminacy in precise timings. The best one can expect is to get some kind of interval indicating how fast or slow each instruction is. For these reasons we abandoned the idea of trying to model precisely the clock interrupt mechanism.

In our model of the BDX930, an interrupt can occur at any time while interrupts are enabled. One must state explicitly where interrupts are assumed not to occur. This exposes a problem in the dispatcher above. If control reaches the dispatcher because of the clock interrupt mechanism, the dispatcher and the scheduler are executed with clock interrupts enabled. A clock interrupt during either of these processes causes chaos. In our model, we must assume explicitly that no clock interrupt occurs during this processing. To prove that no interrupts can occur, one must determine the maximum time it takes to execute the dispatcher and scheduler. To do this one must (a) have a precise specification of the times taken by individual BDX930 operations and (b) treat the scheduler as BDX930 code rather than Pascal. This particular problem could be avoided if the dispatcher always disabled interrupts on entry. However, the complete lack of constraint on interrupts in the current model is unsettling and unrealistic.

Mapping Function: To determine the state of each virtual machine the mapping function must consider each task and determine the state of the task. Consider how one might determine the contents of the thirteen saved registers in each task. The registers of a suspended task are stored in positions 2 through 14 of the stack saved for the task in the task table. The registers of the active task are somewhat more difficult to ascertain. If the program counter (pc) is in the code for the active task, the virtual registers are in the corresponding actual registers. But if the pc is in the dispatcher or scheduler, the virtual registers may be in any number of places. For example, if the instruction at CINT has just been executed, they are in the actual registers. But if the instruction just after CINT has just been executed, the virtual registers are in positions 1 through 13 of the stack in register 15. And if the second instruction after CINT has just been executed, they are in positions 2 through 14 of that stack.

In general, to recover the state of the active task, it is necessary to consider (while defining the mapping function) each instruction in the dispatcher and scheduler. Furthermore, it is necessary to treat the scheduler as BDX930 code rather than as Pascal code, since otherwise one cannot trace where in the real machine the components of the state are being kept while in transit to the task table.

Restrictions on User Tasks: Two restrictions on user tasks are necessary if the dispatcher is to implement the kind of time-sharing system described.

The first concerns the size of the Pascal stack for each task. Recall that the state of an interrupted process is saved by pushing the flags, 13 registers, and the pc on the stack. If there is insufficient room on the stack, instructions or data (possibly from another task) are overwritten. Thus, one restriction on the user tasks is that they never come within 15 words of exhausting the allocated stack space. But the stack is used primarily to store temporaries and subroutine links and its management is entirely under the control of the Pascal compiler. One cannot determine whether a given Pascal program satisfies this restriction unless one looks at the code generated by a given compiler. Note that in general it is impossible to verify with a static analysis that a given user task -- even displayed as BDX930 code -- does not use too much of the stack, since depth of recursion and other runtime considerations influence stack use.

The second restriction is more subtle. In its attempt to save the state of an interrupted process, the dispatcher saves only the flags, registers, and pc. It is assumed that all other parts of the state of the task are private to the task itself and will not be affected by the execution of other tasks. In particular, tasks may not share variables that are read and written. At first sight one may conclude that this assumption can be checked by confirming that the Pascal code for a set of tasks share no variables. However, such a check is insufficient. Again, the compiler must be considered. Suppose that the compiler uses certain memory locations as temporaries. Then those temporaries must be saved by the dispatcher too. But if user tasks are considered to be unrestricted BDX930 code, the check becomes even more difficult because it is not possible to determine with a static analysis what memory locations are read and written. It is also necessary to require of user tasks that they not use the clock interrupt mechanism and not overwrite the area of the BDX930 dedicated to the operating system. Specifying the requirements on user programs requires a rigorous formal understanding of the BDX930, the Pascal compiler, and the linking loader. Thus an attempt to verify the few lines of machine code in SIFT lead to the requirement that we have formal specifications for several huge objects which have never yet been adequately formalized.

This concludes our discussion of difficulties encountered while trying to formalize the simple time-sharing system sketched above. However, the worst is yet to come. The simple model sketched is inadequate for SIFT because tasks are supposed to share data.

It is common in SIFT for task A to compute a result and put it somewhere for a later task, B, to read. For example, many tasks share parts of the datafile with the prevote task. But this suddenly introduces the notion of time. In the simple model, tasks A and B each run on their own processor and do not interract. If each task is to be repeated indefinitely then each processor endlessly iterates its own task. There is no sense in which the iterations of A are synchronized with those of B. Under the current SIFT scheme however, the dispatcher is used to "time share" tasks that share data, but the schedules tables are arranged so that the iterations of A do not overlap those of B.

If one attempts to patch things up while preserving the notion that A and B are running on independent virtual BDX930s, one is forced to introduce the notion of communicating virtual machines -- an idea somewhat more complicated than the truth. We now question the utility of the abstraction of virtual machines. Indeed, the whole idea that the dispatcher is implementing a time-sharing system comes into question since a major use of it is to orchestrate fixed sequences of subroutine calls.

The time-sharing/virtual machine idea is completely destroyed by the reconfiguration task. This tasks redefines the task table. Thus, after termination of the reconfiguration task, the tasks run by the dispatcher have no relation to those run before reconfiguration. It is impossible to view the dispatcher as a time-sharing system implementing virtual BDX930s running concurrently when one "process" can wipe out the others.

In our view, it is a mistake to think of the dispatcher in abstract terms. It seems to be just a program running on a von Neumann machine. By carefully arranging certain tables you can program the machine to execute a few instructions from here and then a few instructions from there, almost as though you had two different machines. By cleverly arranging those tables you can make one piece of code share data with another, almost as though your machines were communicating. By being still more clever you can synchronize them to the point that the two programs appear to be running sequentially on just one machine. Indeed, by carelessly arranging those same tables you can cause arbitrary chaos. But the moral is clear: you are programming a single machine and not a set of virtual machines.

We think that things might be a lot more clear if schedules were not encoded as tables that were interpretted by the scheduler and dispatcher but were merely Pascal programs that iteratively called fixed sequences of subroutines.

Research Topics Worthy of Consideration

Abstract Interpreters:

Interrupts:

Real time -- Newtonian time -- clock synchronization:

SECTION 15

FORMAL DEFINITION OF BDX930

The B930 Adventure

We started with a shell defn for the B930 state which had 16 components, 12 of which were numerically typed. When we ran that through the <BOYER> theorem prover it blew up on the CONS.EQUAL axiom because it tried to normalize the IFs on the rhs of the equality.

We changed DEFNO and MAKE.REWRITE.RULES so that the bodies of defns and the rhs's of rewrites were not altered by the system.

Because we anticipate being wiped out by all the nonrec defns in the B930, we considered editing REWRITE.FNCALL to make it open up nonrec fns more carefully than now. It has been suggested that we adopt some draconian restriction (e.g., open only if no IFs are introduced) just to force us to think about more reasonable restrictions. That would make us fail to prove (AND P Q) -> (AND Q P) by simplification alone. (We could actually prove it, by virtue of the expansion of ANDs in hypotheses if we recognized (AND P Q) as an AND.) Another idea was similar to Woody's pairings (as we have always imagined it) namely keep track of what tests the fns were interested in and have some high level procedure split on the most important ones to force certain fns to open together. In the end we decided not to touch REWRITE.FNCALL for now but it is lying in wait for the B930 to come along.

A BOOT.STRAP failed because DIFFERENCE was no longer numerically typed (and so RECURSION.BY.DIFFERENCE was rejected as an induction lemma) because it returns I after testing not (ZEROP I).

We considered several alternatives. One was the idea of a "type set lemma", e.g., (NOT (ZEROP X)) -> (NUMBERP X), which could be implemented by generalizing RECOGNIZER.ALIST to two alists, one for use when the recognizer is assumed true and one for when it is assumed false. E.g., NLISTP would be bound the type bits for LISTP on the false alist and bound to the complement on the true alist. ZEROP would be on the false one, bound to numbers. This would probably not slow down TYPE.SET. We'd have to write some code to recognize type set lemmas and produce the bit patterns from the recognizer proposition.

A second alternative was to define a new class of functions called "defined recognizers" which were boolean valued nonrec fns and to open them up all the time (in preprocessing) and to normalize. That is the approach we took. We changed PUT.TYPE.PRESCRIPTION to so preprocess sdefns before guessing the type. But we left the unpreprocessed sdefn as the one used in theorem proving.

We then decided to address a problem Bernie raised, namely that (EQUAL NIL NIL) is proved rather circuitously by expanding the abbreviation PACK.EQUAL to the equality of the unpacks and (using CONS.EQUAL under the rule in CLAUSIFY.INPUT that says you can split a conjunction at the top level) rewrote that to the conjoined equalities of the ascii codes. When we tried to reproduce the silly proof with the just modified theorem prover it was not so silly because CONS.EQUAL wasn't used. The reason was that in the modified tp the rhs of CONS.EQUAL is (AND & &) instead of (IF & & F) and so is not recognized as a conjunction!

We are reluctant to let this state of affairs persist because we are basically happy with the current preprocessing of large vcs and don't want to inadvertantly change that preprocessing. We thought perhaps we should always expand ANDs (and other boot strap propositional fns) and normalize. But that blows us out of the water on a 16 component shell.

On the subject of abbreviations, we were troubled by the fact that PACK.EQUAL is an abbreviation but ADD1.EQUAL is not. Thank goodness for that (and the fact that ADD1.EQUAL is not a conjunction), since otherwise (EQUAL 1000 1000) would blow us out of the water a la Bernie. But it seems odd that two schematically related rewrite rules are not treated equally.

Finally, Bernie's problem would never have arisen had the preprocessing put expressions in reduced form. One suggestion is to make CONS.TERM always apply lfns, instead of just doing it for shells. If we did that, one might argue that we ought to review the uses of FCONS.TERM to determine whether they should be replaced by CONS.TERMs to enforce a new invariant on terms, namely that they are always in reduced form.

Another subject we have discussed is that we should make nonrecursive functions and unconditional rewrite rules behave identically. This idea was suggested after considering further how we might handle a 16 component shell with type restrictions. E.g., we could make CONS.EQUAL rewrite (EQUAL (CONS X Y) (CONS U V)) to the conjunction of things like

(EQUAL (CAR (CONS X Y)) (CAR (CONS U V))).

And then treat the rewrite rule (CAR (CONS X Y)) = (IF type X dv) in the way we treat nonrec defns, namely not apply such rewrites when they introduce too many IFs or otherwise blow us up.

If we decided to make unconditional rewrites and nonrec fns behave identically we could do it by eliminating nonrec defns altogether and just storing them as rewrite rules.

Another idea on the subject of large shells is that we could rewrite the equality of two conses to the equality of some coercions, eg. (FIX X) = (FIX U), instead of naked IFs, and then let the nonrec fn handler worry about the explosion.

Still another suggestion was to eliminate shells other than the boot strap ones and force people to use lists the way mathematicians do. That will probably require some better handling of nonrec fns than we have now since abbreviations become quite useful. There is also the worry that things will get sticky the way they did in Edinburgh when we couldn't distinguish lists from numbers.

```
(SETQ XXX '(
(ADD.SHELL STATE NIL STATEP
   ((MEM (NONE.OF) ZERO)
    (PC (NONE.OF) ZERO)
    (ACS (NONE.OF) ZERO)
    (OV (NONE.OF) ZERO)
    (IE (NONE.OF) ZERO)
    (IR (NONE.OF) ZERO)
    (F1 (NONE.OF) ZERO)
    (F2 (NONE.OF) ZERO)
    (EXT1 (NONE.OF) ZERO)
    (EXT2 (NONE.OF) ZERO)
    (EXT3 (NONE.OF) ZERO)
    (HALT (NONE.OF) FALSE)
    (ERROR (NONE.OF) FALSE)
    (CONTROL.PANELP (NONE.OF) FALSE)
    (INDIRECT.CNT (NONE.OF) ZERO)
    (EXECR.CNT (NONE.OF) ZERO)))

(DEFN FETCH (MEM LOC)
    (IF (NLISTP MEM) 0
        (IF (EQUAL (CAAR MEM) LOC) (CDAR MEM) (FETCH (CDR MEM) LOC))))
(DEFN PUT (LOC VAL MEM) (CONS (CONS LOC VAL) MEM))

(DEFN SET.MEM (LOC VAL ST)
      (STATE (PUT LOC VAL (MEM ST))
             (PC ST)
             (ACS ST)
             (OV ST)
             (IE ST)
             (IR ST)
             (F1 ST)
             (F2 ST)
             (EXT1 ST)
             (EXT2 ST)
             (EXT3 ST)
             (HALT ST)
             (ERROR ST)
             (CONTROL.PANELP ST)
             (INDIRECT.CNT ST)
             (EXECR.CNT ST)))

(DEFN SET.PC (PC ST)
      (STATE (MEM ST)
             PC
             (ACS ST)
             (OV ST)
             (IE ST)
             (IR ST)
             (F1 ST)
             (F2 ST)
             (EXT1 ST)
             (EXT2 ST)
             (EXT3 ST)
             (HALT ST)
             (ERROR ST)
             (CONTROL.PANELP ST)
             (INDIRECT.CNT ST)
             (EXECR.CNT ST)))
```

```
(DEFN SET.AC (AC VAL ST)
      (STATE (MEM ST)
             (PC ST)
             (PUT AC VAL (ACS ST))
             (OV ST)
             (IE ST)
             (IR ST)
             (F1 ST)
             (F2 ST)
             (EXT1 ST)
             (EXT2 ST)
             (EXT3 ST)
             (HALT ST)
             (ERROR ST)
             (CONTROL.PANELP ST)
             (INDIRECT.CNT ST)
             (EXECR.CNT ST)))

(DEFN SET.AC.&.AC+1 (AC PAIR ST)
      (STATE (MEM ST)
             (PC ST)
             (PUT AC (CAR PAIR) (PUT (ADD1 AC) (CDR PAIR)  (ACS ST)))
             (OV ST)
             (IE ST)
             (IR ST)
             (F1 ST)
             (F2 ST)
             (EXT1 ST)
             (EXT2 ST)
             (EXT3 ST)
             (HALT ST)
             (ERROR ST)
             (CONTROL.PANELP ST)
             (INDIRECT.CNT ST)
             (EXECR.CNT ST)))

(DEFN SET.OV (VAL ST)
      (STATE (MEM ST)
             (PC ST)
             (ACS ST)
             VAL
             (IE ST)
             (IR ST)
             (F1 ST)
             (F2 ST)
             (EXT1 ST)
             (EXT2 ST)
             (EXT3 ST)
             (HALT ST)
             (ERROR ST)
             (CONTROL.PANELP ST)
             (INDIRECT.CNT ST)
             (EXECR.CNT ST)))

(DEFN SET.IE (VAL ST)
      (STATE (MEM ST)
             (PC ST)
             (ACS ST)
```

```
                (OV ST)
                VAL
                (IR ST)
                (F1 ST)
                (F2 ST)
                (EXT1 ST)
                (EXT2 ST)
                (EXT3 ST)
                (HALT ST)
                (ERROR ST)
                (CONTROL.PANELP ST)
                (INDIRECT.CNT ST)
                (EXECR.CNT ST)))


(DEFN SET.F1 (VAL ST)
      (STATE (MEM ST)
             (PC ST)
             (ACS ST)
             (OV ST)
             (IE ST)
             (IR ST)
             VAL
             (F2 ST)
             (EXT1 ST)
             (EXT2 ST)
             (EXT3 ST)
             (HALT ST)
             (ERROR ST)
             (CONTROL.PANELP ST)
             (INDIRECT.CNT ST)
             (EXECR.CNT ST)))
(DEFN SET.F2 (VAL ST)
      (STATE (MEM ST)
             (PC ST)
             (ACS ST)
             (OV ST)
             (IE ST)
             (IR ST)
             (F1 ST)
             VAL
             (EXT1 ST)
             (EXT2 ST)
             (EXT3 ST)
             (HALT ST)
             (ERROR ST)
             (CONTROL.PANELP ST)
             (INDIRECT.CNT ST)
             (EXECR.CNT ST)))
(DEFN SET.HALT (VAL ST)
      (STATE (MEM ST)
             (PC ST)
             (ACS ST)
             (OV ST)
             (IE ST)
             (IR ST)
             (F1 ST)
             (F2 ST)
```

```
                (EXT1 ST)
                (EXT2 ST)
                (EXT3 ST)
                VAL
                (ERROR ST)
                (CONTROL.PANELP ST)
                (INDIRECT.CNT ST)
                (EXECR.CNT ST)))
(DEFN SET.ERROR (VAL ST)
      (STATE (MEM ST)
                (PC ST)
                (ACS ST)
                (OV ST)
                (IE ST)
                (IR ST)
                (F1 ST)
                (F2 ST)
                (EXT1 ST)
                (EXT2 ST)
                (EXT3 ST)
                (HALT ST)
                VAL
                (CONTROL.PANELP ST)
                (INDIRECT.CNT ST)
                (EXECR.CNT ST)))



(DEFN EXPT (I J)
      (IF (ZEROP J) 1 (TIMES I (EXPT I (SUB1 J)))))

(DEFN FIELD (WRD HI LO) (REMAINDER (QUOTIENT WRD (EXPT 2 LO))
                                   (EXPT 2 (ADD1 (DIFFERENCE HI LO)))))



                        (* to be defined to return the integer
                          represented by bits HI through LO
                          inclusive in the binary representation
                          of the integer WRD.))

(DEFN AM (WRD) (FIELD WRD 11 10))
(DEFN IBIT (WRD) (FIELD WRD 15 15))
(DEFN D (WRD) (FIELD WRD 7 0))
(DEFN OP1 (WRD) (FIELD WRD 14 12))
(DEFN OP2 (WRD)(FIELD WRD 11 8))
(DEFN A (WRD) (FIELD WRD 7 4))
(DEFN B (WRD) (FIELD WRD 3 0))
(DEFN AC (WRD) (FETCH WRD 9 8))
(DEFN OP3 (WRD) (FIELD WRD 7 4))
(DEFN DELTA (WRD) (FETCH WRD 3 0))

(DEFN TURN.OFF.HI.BIT (WRD) (FIELD WRD 14 0))

(DCL B930.ADD.8BIT (WRD DISPL)
     (* adds the 16 bit quantity WRD to the 8 bit
        signed quantity DISPL and produces a new 16 bit
        quantity.  This fn is always used to
        construct an address -- e.g., a pc or
```

```
        stack pointer or effective address.
        We are not sure what happens if WRD represents
        a negative quantity.  Also, we assume such
        arithmetic isn't senstive to the F1 flg.  Also
        we don't know what happens when an overflow occurs.))

(DCL B930.ADD.4BIT (WRD DELTA)
     (* adds 16 bits to 4 bit signed quantity to
        produce new 16 bit thing.  The comments about
        B930.ADD.8BIT apply here too.))

(DEFN PC+1 (ST) (B930.ADD.8BIT (PC ST) 1))

(DEFN TRACE.INDIRECT.CHAIN (WRD MEM CNT)
 (IF (ZEROP CNT)
     0
     (IF (EQUAL (IBIT WRD) 1)
         (TRACE.INDIRECT.CHAIN (FETCH MEM (TURN.OFF.HI.BIT WRD))
                               MEM (SUB1 CNT))
         WRD))
 (* We suppose you turn the high bit off before you treat WRD as
    an address. The ISP doesn't.))



(DEFN MAR (WRD ST)
 (TRACE.INDIRECT.CHAIN
  (IF (EQUAL (AM WRD) 0)
      (D WRD)
  (IF (EQUAL (AM WRD) 1)
      (B930.ADD.8BIT (PC ST) (D WRD))
  (IF (EQUAL (AM WRD) 2)
      (B930.ADD.8BIT (FETCH (ACS ST) 0)
                     (D WRD))
      (B930.ADD.8BIT (FETCH (ACS ST) 1)
                     (D WRD)))))
  (MEM ST) (INDIRECT.CNT ST))
 (* The ISP seems to just add the displacement when we think it
    ought to use the 8-bit add and permit negative displacements.
    The ISP indicates that the indirect bit is to be interpreted
    as here, i.e., once calculate an address from D etc and then
    chain through the indirect pointers.  But the programmers manual
    has evidence that the MAR calculation is more akin to the PDP-10
    style where one recomputes the effective address recursively.
    Every place we call MAR on WRD2 of a double word instr, we pass
    an ST whose PC points to the first of the two words.  Should it
    pass BUMP.PC of ST instead?  The ISP indicates yes with its
    GROUP command, but the manual indicates no under the discussion
    of JSS.))
(DCL B930.ADDR (WRD1 WRD2 F1) (* Returns the 16 bit number the
                                 B930 would if asked to add
                                 WRD1 and WRD2 with F1 set to 1 or 0.))
(DCL B930.SUBR (WRD1 WRD2 F1))

(DCL B930.ADDR.OV (WRD1 WRD2 F1) (* Returns value of OV flag after
                                    the appropriate B930 add.  What
                                    is the parity of the bit?  Sometimes
                                    -- e.g. in DECEQ -- we set the
```

```
                                        OV flg to 0 meaning NO OVERFLOW. Is
                                        that right?))
(DCL B930.SUBR.OV (WRD1 WRD2 F1))


(DEFN JU (WRD1 WRD2 ST) (SET.PC (MAR WRD1 ST) ST))
(DEFN JSA0 (WRD1 WRD2 ST)
          (SET.PC (MAR WRD1 ST)
                  (SET.AC 0 (PC+1 ST) ST))
          (* We compute MAR w.r.t. the unmodified state.  But the
             manual and the ISP imply that MAR is computed after
             smashing ac 0; We did it this way just because it
             seemed more likely.))
(DEFN JSA1 (WRD1 WRD2 ST)
      (SET.PC (MAR WRD1 ST)
              (SET.AC 1 (PC+1 ST) ST))
      (* See JSA0))
(DEFN JMA0  (WRD1 WRD2 ST)
 (SET.PC (MAR WRD1 ST)
         (SET.AC 0 (PC+1 ST)
                 (SET.MEM (B930.ADD.8BIT (FETCH (ACS ST) 15) 1)
                          (FETCH (ACS ST) 0)
                          (SET.AC 15 (B930.ADD.8BIT (FETCH (ACS ST) 15) 1)
                                     ST))))
      (* we are unsure of whether we are to compute
         MAR of the original ST as here or of
         state after the modifications below.  Note that the treatment
         of indirect address chains is affected.))
(DEFN ADD  (WRD1 WRD2 ST)
 (BUMP.PC (SET.AC (AC WRD1)
                  (B930.ADDR (FETCH (ACS ST) (AC WRD1))
                             (FETCH (MEM ST) (MAR WRD1 ST))
                             (F1 ST))
                  (SET.OV (B930.ADDR.OV
                           (FETCH (ACS ST) (AC WRD1))
                           (FETCH (MEM ST) (MAR WRD1 ST))
                           (F1 ST))
                          ST))))
(DEFN SUB  (WRD1 WRD2 ST)
 (BUMP.PC (SET.AC (AC WRD1)
                  (B930.SUBR (FETCH (ACS ST) (AC WRD1))
                             (FETCH (MEM ST) (MAR WRD1 ST))
                             (F1 ST))
                  (SET.OV (B930.SUBR.OV
                           (FETCH (ACS ST) (AC WRD1))
                           (FETCH (MEM ST) (MAR WRD1 ST))
                           (F1 ST))
                          ST))))
(DEFN CMP  (WRD1 WRD2 ST)
 (SET.PC
  (B930.ADD.8BIT (PC ST)
                 (IF (B930.LESSP (FETCH (ACS ST)
                                        (AC WRD1))
                                 (FETCH (MEM ST)
                                        (MAR WRD1 ST)))
                     3
                     (IF (B930.EQP (FETCH (ACS ST) (AC WRD1))
                                   (FETCH (MEM ST) (MAR WRD1 ST)))
```

```
                          1
                          2)))
  (SET.OV (B930.CMP.OV WRD1 WRD2 ST) ST)))
(DEFN LOAD  (WRD1 WRD2 ST)
        (BUMP.PC (SET.AC (AC WRD1)
                         (FETCH (MEM ST) (MAR WRD1 ST))
                         ST)))
(DEFN STO  (WRD1 WRD2 ST)
        (BUMP.PC (SET.MEM (MAR WRD1 ST)
                          (FETCH (ACS ST) (AC WRD1))
                          ST)))
(DEFN TRA/NOP (WRD1 WRD2 ST)
        (BUMP.PC (SET.AC (A WRD1)
                         (FETCH (ACS ST) (B WRD1))
                         ST)))
(DEFN DECEQ  (WRD1 WRD2 ST)
 (IF (EQUAL (FETCH (ACS ST) (A WRD1)) 1)
     (SET.PC
      (B930.ADD.4BIT (PC+1 ST)
                     (B WRD1))
      (SET.AC (A WRD1) 0
              (SET.OV 0 ST)))
     (BUMP.PC (SET.AC (A WRD1)
                      (B930.SUBR (FETCH (ACS ST) (A WRD1))
                                 1
                                 (F1 ST))
                      (SET.OV (B930.SUBR.OV (FETCH (ACS ST)
                                                   (A WRD1))
                                            1
                                            (F1 ST))
                              ST)))))
(DEFN LCM  (WRD1 WRD2 ST)
        (BUMP.PC (SET.AC (A WRD1)
                         (B930.LCM (FETCH (ACS ST)
                                          (B WRD1)))
                         ST)))

(DEFN RLS  (WRD1 WRD2 ST)
        (BUMP.PC (SET.AC (A WRD1)
                         (B930.RLS (FETCH (ACS ST) (A WRD1))
                                   (B WRD1))
                         ST)))
(DEFN B930.CONT (OLD FN)
 (IF (EQUAL FN 0) OLD
     (IF (EQUAL FN 1) 0
         (IF (EQUAL FN 2) 1 (IF (EQUAL OLD 0) 1 0)))))

(DEFN CONT  (WRD1 WRD2 ST)
 (BUMP.PC
  (SET.F1 (B930.CONT (F1 ST) (FIELD WRD1 5 4))
          (SET.F2 (B930.CONT (F2 ST) (FIELD WRD1 7 6))
                  (SET.IE (B930.CONT (IE ST) (FIELD WRD1 3 2))
                          (SET.OV (B930.CONT (OV ST) (FIELD WRD1 1 0)) ST))))))



(DEFN DECNE  (WRD1 WRD2 ST)
 (IF (NOT (EQUAL (FETCH (ACS ST) (A WRD1)) 1))
     (SET.PC
```

```
      (B930.ADD.4BIT (PC+1 ST)
                     (B WRD1))
      (SET.AC (A WRD1)
              (B930.SUBR (FETCH (ACS ST) (A WRD1))
                         1
                         (F1 ST))
              (SET.OV (B930.SUBR.OV (FETCH (ACS ST)
                                           (A WRD1))
                                    1
                                    (F1 ST))
                      ST)))
    (BUMP.PC (SET.AC (A WRD1) 0
                     (SET.OV 0 ST)))))
(DEFN ANDOP (WRD1 WRD2 ST)
 (BUMP.PC (SET.AC (A WRD1)
                  (B930.AND (FETCH (ACS ST) (A WRD1))
                            (FETCH (ACS ST) (B WRD1)))
                  ST)))
(DEFN RLL  (WRD1 WRD2 ST)
        (IF (NOT (EVEN (A WRD1)))
            (SET.ERROR T ST)
            (BUMP.PC
             (SET.AC.&.AC+1 (A WRD1)
                            (B930.RLL (FETCH (ACS ST)
                                             (A WRD1))
                                      (FETCH (ACS ST)
                                             (ADD1 (A WRD1)))
                                      (B WRD1))
                            ST))))
(DEFN ADDR  (WRD1 WRD2 ST)
 (BUMP.PC
  (SET.AC (A WRD1)
          (B930.ADDR (FETCH (ACS ST) (A WRD1))
                     (FETCH (ACS ST) (B WRD1))
                     (F1 ST))
          (SET.OV (B930.ADDR.OV
                   (FETCH (ACS ST) (A WRD1))
                   (FETCH (ACS ST) (B WRD1))
                   (F1 ST))
                  ST))))
(DEFN IR/CLA (WRD1 WRD2 ST)
 (IF (EQUAL (A WRD1) (B WRD1))
     (BUMP.PC (SET.AC (A WRD1) 0 ST))
     (BUMP.PC
      (SET.AC (A WRD1)
              (FETCH (ACS ST) (B WRD1))
              (SET.AC (B WRD1)
                      (FETCH (ACS ST) (A WRD1))
                      ST))))
  (* The ISP uses arithmetic to achieve this effect.  The manual
     does not mention the use of arithmetic.  Neither document
     suggests that OV gets set.)
)

(DEFN OROP (WRD1 WRD2 ST)
 (BUMP.PC (SET.AC (A WRD1)
                  (B930.OR (FETCH (ACS ST) (A WRD1))
                           (FETCH (ACS ST) (B WRD1)))
                  ST)))
```

```
(DCL B930.MPY (A B) (* Returns the double word value of A times B.
                       The manual and the ISP both suggest that something
                       weird happens at -1 but nobody seems to know what
                       it is.  The manual suggests that something weird
                       happens with the most negative number.  Finally,
                       the result is apparently left shifted one.))
(DEFN MPY  (WRD1 WRD2 ST)
 (IF (OR (NOT (EVEN (A WRD1)))
         (EQUAL (A WRD1) (B WRD1))
         (EQUAL (ADD1 (A WRD1)) (B WRD1)))
     (SET.ERROR T ST)
     (BUMP.PC
      (SET.AC.&.AC+1 (A WRD1)
                     (B930.MPY (FETCH (ACS ST) (A WRD1))
                               (FETCH (ACS ST) (B WRD1)))
                     ST))))
(DEFN CLAO/SUBR (WRD1 WRD2 ST)
 (BUMP.PC
  (SET.AC (A WRD1)
          (B930.SUBR (FETCH (ACS ST) (A WRD1))
                     (FETCH (ACS ST) (B WRD1))
                     (F1 ST))
          (SET.OV (B930.SUBR.OV
                   (FETCH (ACS ST) (A WRD1))
                   (FETCH (ACS ST) (B WRD1))
                   (F1 ST))
                  ST)))
          (* The ISP says OV gets zeroed but we think not.))

(DEFN ACM  (WRD1 WRD2 ST)
 (BUMP.PC
  (SET.AC (A WRD1)
          (B930.SUBR 0
                     (FETCH (ACS ST) (B WRD1))
                     (F1 ST))
          (SET.OV (B930.SUBR.OV
                   0
                   (FETCH (ACS ST) (B WRD1))
                   (F1 ST))
                  ST))))
(DEFN CMPR  (WRD1 WRD2 ST)
 (SET.PC (B930.ADD.8BIT (PC ST)
                        (IF (B930.LESSP (FETCH (ACS ST)
                                               (A WRD1))
                                        (FETCH (ACS ST)
                                               (B WRD1)))
                            3
                            (IF (B930.EQP (FETCH (ACS ST) (A WRD1))
                                          (FETCH (ACS ST) (B WRD1)))
                                1
                                2)))
         (SET.OV (B930.CMP.OV WRD1 WRD2 ST) ST)))
(DEFN DIV  (WRD1 WRD2 ST)
         (IF (OR (NOT (EVEN (A WRD1)))
                 (EQUAL (A WRD1) (B WRD1))
                 (EQUAL (ADD1 (A WRD1)) (B WRD1)))
             (SET.ERROR T ST)
             (BUMP.PC
              (SET.AC.&.AC+1 (A WRD1)
```

```
                                (B930.DIV (FETCH (ACS ST) (A WRD1))
                                          (FETCH (ACS ST) (ADD1 (A WRD1)))
                                          (FETCH (ACS ST) (B WRD1)))
                                (SET.OV
                                 (B930.DIV.OV
                                  (FETCH (ACS ST) (A WRD1))
                                  (FETCH (ACS ST) (ADD1 (A WRD1)))
                                  (FETCH (ACS ST) (B WRD1)))
                                 ST))))
          (* The ISP contains an error in that SP[A] instead of SP[A]@SP[A+1] is
             compared with SP[B])
)
(DEFN SLSA  (WRD1 WRD2 ST)
 (BUMP.PC (SET.AC (A WRD1)
                  (B930.SLSA (FETCH (ACS ST) (A WRD1))
                             (B WRD1))
                  (SET.OV (B930.SLSA.OV (FETCH (ACS ST)(A WRD1))
                                        (B WRD1))
                          ST))))
(DEFN SLLA  (WRD1 WRD2 ST)
 (IF (NOT (EVEN (A WRD1)))
     (SET.ERROR T ST)
     (BUMP.PC
      (SET.AC.&.AC+1 (A WRD1)
                     (B930.SLLA (FETCH (ACS ST)
                                       (A WRD1))
                                (FETCH (ACS ST)
                                       (ADD1 (A WRD1)))
                                (B WRD1))
                     (SET.OV
                      (B930.SLLA (FETCH (ACS ST)
                                        (A WRD1))
                                 (FETCH (ACS ST)
                                        (ADD1 (A WRD1)))
                                 (B WRD1))
                      ST)))))
(DEFN SKGT  (WRD1 WRD2 ST)
 (IF (B930.LESSP 0 (FETCH (ACS ST) (A WRD1)))
     (SET.PC
      (B930.ADD.4BIT (PC+1 ST)
                     (B WRD1))
      ST)
     (BUMP.PC ST)))
(DEFN SKLT  (WRD1 WRD2 ST)
 (IF (B930.LESSP (FETCH (ACS ST) (A WRD1)) 0)
     (SET.PC
      (B930.ADD.4BIT (PC+1 ST)
                     (B WRD1))
      ST)
     (BUMP.PC ST)))
(DEFN SLSL  (WRD1 WRD2 ST)
 (BUMP.PC (SET.AC (A WRD1)
                  (B930.SLSL (FETCH (ACS ST) (A WRD1))
                             (B WRD1))
                  ST)))
(DEFN SLLL  (WRD1 WRD2 ST)
 (IF (NOT (EVEN (A WRD1)))
      (SET.ERROR T ST)
     (BUMP.PC
```

```
      (SET.AC.&.AC+1 (A WRD1)
                     (B930.SLLL (FETCH (ACS ST)
                                       (A WRD1))
                                (FETCH (ACS ST)
                                       (ADD1 (A WRD1)))
                                (B WRD1))
                     ST))))
(DEFN SKGE  (WRD1 WRD2 ST)
 (IF (NOT (B930.LESSP (FETCH (ACS ST) (A WRD1)) 0))
     (SET.PC
      (B930.ADD.4BIT (PC+1 ST)
                     (B WRD1))
      ST)
     (BUMP.PC ST)))
(DEFN SKLE  (WRD1 WRD2 ST)
 (IF (NOT (B930.LESSP 0 (FETCH (ACS ST) (A WRD1))))
     (SET.PC
      (B930.ADD.4BIT (PC+1 ST)
                     (B WRD1))
      ST)
     (BUMP.PC ST)))
(DEFN SRSA  (WRD1 WRD2 ST)
 (BUMP.PC (SET.AC (A WRD1)
                  (B930.SRSA (FETCH (ACS ST) (A WRD1))
                             (B WRD1))
                  ST)))
(DEFN SRLA  (WRD1 WRD2 ST)
  (IF (NOT (EVEN (A WRD1)))
      (SET.ERROR T ST)
      (BUMP.PC
       (SET.AC.&.AC+1 (A WRD1)
                      (B930.SRLA (FETCH (ACS ST)
                                        (A WRD1))
                                 (FETCH (ACS ST)
                                        (ADD1 (A WRD1)))
                                 (B WRD1))
                      ST))))
(DEFN SKEQ  (WRD1 WRD2 ST)
 (IF (B930.EQP 0 (FETCH (ACS ST) (A WRD1)))
     (SET.PC
      (B930.ADD.4BIT (PC+1 ST)
                     (B WRD1))
      ST)
     (BUMP.PC ST)))
(DEFN SRSL  (WRD1 WRD2 ST)
 (BUMP.PC (SET.AC (A WRD1)
                  (B930.SRSL (FETCH (ACS ST) (A WRD1))
                             (B WRD1))
                  ST)))
(DEFN SRLL  (WRD1 WRD2 ST)
 (IF (NOT (EVEN (A WRD1)))
     (SET.ERROR T ST)
     (BUMP.PC
      (SET.AC.&.AC+1 (A WRD1)
                     (B930.SRLL (FETCH (ACS ST)
                                       (A WRD1))
                                (FETCH (ACS ST)
                                       (ADD1 (A WRD1)))
                                (B WRD1))
```

```
                        ST))))
(DEFN SKNE  (WRD1 WRD2 ST)
 (IF (NOT (B930.EQP 0 (FETCH (ACS ST) (A WRD1))))
     (SET.PC
      (B930.ADD.4BIT (PC+1 ST) (B WRD1))
      ST)
     (BUMP.PC ST)))

(DEFN IAR  (WRD1 WRD2 ST)
 (BUMP.PC (SET.AC (A WRD1)
                  (B930.ADDR (FETCH (ACS ST) (A WRD1))
                             (IF (LESSP (B WRD1) 8)
                                 (B WRD1)
                                 (DIFFERENCE
                                  (PLUS (EXPT 2 15)
                                        (B WRD1))
                                  (EXPT 2 3)))
                             (F1 ST))
                  (SET.OV (B930.ADDR.OV (FETCH (ACS ST) (A WRD1))
                                        (IF (LESSP (B WRD1) 8)
                                            (B WRD1)
                                            (DIFFERENCE
                                             (PLUS (EXPT 2 15)
                                                   (B WRD1))
                                             (EXPT 2 3)))
                                        (F1 ST))))))



(DEFN SFE1 (WRD1 WRD2 ST)
      (IF (EQUAL (EXT1 ST) 0)
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
          (BUMP.PC ST)) (* We have assumed that TRUE means 1 and FALSE 0))

(DEFN SFE2 (WRD1 WRD2 ST)
      (IF (EQUAL (EXT2 ST) 0)
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
          (BUMP.PC ST)))

(DEFN SFE3 (WRD1 WRD2 ST)
      (IF (EQUAL (EXT3 ST) 0)
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
          (BUMP.PC ST)))

(DEFN SRIE (WRD1 WRD2 ST)
      (IF (EQUAL (IE ST) 0)
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
          (BUMP.PC ST))
```

```
      (* The ISP calls this instr SFIE))

(DEFN SROV (WRD1 WRD2 ST)
      (IF (EQUAL (OV ST) 0)
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
          (BUMP.PC ST)))
(DEFN SFIR (WRD1 WRD2 ST)
      (IF (EQUAL (IR ST) 0)
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
          (BUMP.PC ST)))
(DEFN SRF1 (WRD1 WRD2 ST)
      (IF (EQUAL (F1 ST) 0)
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
          (BUMP.PC ST)))
(DEFN SRF2 (WRD1 WRD2 ST)
      (IF (EQUAL (F2 ST) 0)
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
          (BUMP.PC ST)))

(DEFN STE1 (WRD1 WRD2 ST)
      (IF (NOT(EQUAL (EXT1 ST) 0))
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
          (BUMP.PC ST)))

(DEFN STE2 (WRD1 WRD2 ST)
      (IF (NOT(EQUAL (EXT2 ST) 0))
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
          (BUMP.PC ST)))

(DEFN STE3 (WRD1 WRD2 ST)
      (IF (NOT(EQUAL (EXT3 ST) 0))
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
          (BUMP.PC ST)))

(DEFN SSIE (WRD1 WRD2 ST)
      (IF (NOT(EQUAL (IE ST) 0))
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
          (BUMP.PC ST)))
(DEFN SSOV (WRD1 WRD2 ST)
      (IF (NOT(EQUAL (OV ST) 0))
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
```

```
          (BUMP.PC ST)))
(DEFN STIR (WRD1 WRD2 ST)
      (IF (NOT(EQUAL (IR ST) 0))
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
          (BUMP.PC ST)))
(DEFN SSF1 (WRD1 WRD2 ST)
      (IF (NOT(EQUAL (F1 ST) 0))
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
          (BUMP.PC ST)))
(DEFN SSF2 (WRD1 WRD2 ST)
      (IF (NOT(EQUAL (F2 ST) 0))
          (SET.PC (B930.ADD.4BIT (PC+1 ST)
                                 (B WRD1))
                  ST)
          (BUMP.PC ST)))


(DEFN SET.MULTIPLE.ACS (AC ADDR N ST)
 (IF (ZEROP N)
     ST
     (IF (LESSP AC 16)
         (SET.MULTIPLE.ACS (ADD1 AC)
                           (ADD1 ADDR)
                           (SUB1 N)
                           (SET.AC AC (FETCH (MEM ST) ADDR) ST))
         (SET.ERROR T ST)))
 (* We assume that if asked to smash acs beyond 15 we cause
    an error, but will have modified the preceding acs.
    We don't consider the possibility that ADDR is pushed
    beyond 15 bits.))

(DEFN LDM (WRD1 WRD2 ST)
 (SET.PC (B930.ADD.8BIT (PC ST) 2)
         (SET.MULTIPLE.ACS (A WRD1)
                           (MAR WRD2 ST)
                           (ADD1 (DELTA WRD1))
                           ST))
 (* Is the MAR to be calculated each iteration as in the ISP or just
    once as we have done?  The manual says that B+1 consecutive memory
    words are moved -- which agrees with us.
    Also, when calculating the MAR of the second
    word of an instruction, does the PC point to the first or second
    word?  We assume the second.  What if the effective address is
    eventually bumped beyond the end of memory?))

(DEFN SET.MULTIPLE.MEM (AC ADDR N ST)
 (IF (ZEROP N)
     ST
     (IF (LESSP AC 16)
         (STORE.MULTIPLE.MEM (ADD1 AC)
                             (ADD1 ADDR)
                             (SUB1 N)
                             (SET.MEM ADDR (FETCH (ACS ST) AC) ST))
         (SET.ERROR T ST)))
 (* We assume that if asked to access acs beyond 15 we cause
```

```
   an error, but will have modified the preceding acs.
   We don't consider the possibility that ADDR is pushed
   beyond 15 bits.))

(DEFN STM (WRD1 WRD2 ST)
 (SET.PC (B930.ADD.8BIT (PC ST) 2)
         (SET.MULTIPLE.MEM (A WRD1)
                           (MAR WRD2 ST)
                           (ADD1 (DELTA WRD1))
                           ST))
 (* See the questions under LDM))

(DEFN PADDM/POPM/PUSHM (WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD2) 0)
     (PADDM WRD1 WRD2 ST)
     (IF (EQUAL (OP2 WRD2) 1)
         (POPM WRD1 WRD2 ST)
         (IF (EQUAL (OP2 WRD2) 2)
             (PUSHM WRD1 WRD2 ST)
             (SET.ERROR T ST)))))

(DEFN POPM (WRD1 WRD2 ST)
 (IF (OR (LESSP (FETCH (ACS ST) (B WRD2))
                (ADD1 (DELTA WRD1)))
         (LESSP (A WRD2) (DELTA WRD1)))
     (SET.ERROR T ST)
     (SET.PC (B930.ADD.8BIT (PC ST) 2)
             (SET.AC (B WRD2)
                     (DIFFERENCE (FETCH (ACS ST) (B WRD2))
                                 (ADD1 (DELTA WRD1)))
                     (SET.MULTIPLE.ACS (DIFFERENCE (A WRD2) (DELTA WRD1))
                                       (DIFFERENCE (FETCH (ACS ST) (B WRD2))
                                                   (DELTA WRD1))
                                       (ADD1 (DELTA WRD1))
                                       ST)))))
 (* We don't know what happens if the initial A is too small to
    be decremented delta+1 times.  We don't know what happens if
    the stack pointer in B is negative or too small to be popped
    delta+1 times.  We cause errors.  We assume the ISP is wrong
    when it says you go back to the stack pointer in B each time
    -- permitting it to be one of the acs smashed -- instead of
    just moving delta+1 consecutive words as stated by the manual.))

(DEFN PUSHM (WRD1 WRD2 ST)
 (IF (OR (NOT (LESSP (IPLUS (FETCH (ACS ST) (B WRD2)) (ADD1 (DELTA WRD1)))
                     (EXPT 2 16)))
         (NOT (LESSP (IPLUS (A WRD2) (DELTA WRD1)) 16)))
     (SET.ERROR T ST)
     (SET.PC (B930.ADD.8BIT (PC ST) 2)
             (SET.AC (B WRD2)
                     (IPLUS (FETCH (ACS ST) (B WRD2)) (ADD1 (DELTA WRD1)))
                     (SET.MULTIPLE.MEM (A WRD2)
                                       (ADD1 (FETCH (ACS ST) (B WRD2)))
                                       (ADD1 (DELTA WRD1))
                                       ST)))))
 (* see the comments under POPM))
```

```
(DEFN EXOR (WRD1 WRD2 ST)
      (BUMP.PC (SET.AC (A WRD1)
                       (B930.EXOR (FETCH (ACS ST)
                                         (A WRD1))
                                  (FETCH (ACS ST)
                                         (B WRD1)))
                       ST)))

(DEFN HALT (WRD1 WRD2 ST)
   (IF (CONTROL.PANELP ST)
       (BUMP.PC (SET.HALT T ST))
       (BUMP.PC ST)))

(DEFN RET (WRD1 WRD2 ST)
 (IF (ZEROP (FETCH (ACS ST) 15))
     (SET.ERROR T ST)
     (SET.PC (B930.ADD.8BIT (FETCH (ACS ST) 0) (D WRD1))
             (SET.AC 0 (FETCH (MEM ST) (FETCH (ACS ST) 15))
                     (SET.AC 15 (SUB1 (FETCH (ACS ST) 15))
                                ST)))
 (* We don't know if the new PC is supposed to permit a negative
    displacement or not, i.e., whether we can really use our
    8-bit adder. The ISP is wrong because it doesn't reference
    memory, it just loads ac 15 into ac 0)))

(DEFN JSS (WRD1 WRD2 ST)
 (SET.PC (MAR WRD2 ST)
         (SET.MEM (B930.ADD.8BIT (FETCH (ACS ST) 15) 1)
                  (B930.ADD.8BIT (PC ST) 2)
                  (SET.AC 15
                          (B930.ADD.8BIT (FETCH (ACS ST) 15) 1)
                          ST)))
 (* We don't really know the order of things.  Is the stack
    smashed and the stack pointer bumped before or after the MAR
    calculation?  Another question concerns the right half of
    the first word of the instr.  The manual does not specify
    whether those bits are important or not.  Our dispatcher,
    EXEC17, treats them as though the manual said they were
    don't cares.))


(DEFN RPS (WRD1 WRD2 ST)
 (IF (ZEROP (FETCH (ACS ST) 15))
     (SET.ERROR T ST)
     (SET.PC (B930.ADD.8BIT (FETCH (MEM ST)
                                   (FETCH (ACS ST) 15))
                            (D WRD2))
             (SET.AC 15
                     (SUB1 (FETCH (ACS ST) 15))
                     ST)))
 (* Is the right half of the first word of the instr important?
    Our EXEC17 treates it as don't care.))

(DEFN POPF (WRD1 WRD2 ST)
  (IF (ZEROP (FETCH (ACS ST) (B WRD2)))
      (SET.ERROR T ST)
      (SET.PC (B930.ADD.8BIT (PC ST) 2)
              (SET.SW (FETCH (MEM ST) (FETCH (ACS ST) (B WRD2)))
                      (SET.AC (B WRD2)
```

```
                                (SUB1 (FETCH (ACS ST) (B WRD2)))
                                ST))))
  (* What is the difference between the "status word" of the
     programmers manual and the "switch register" of the ISP?
     Is the manual setting of switches and POPF the only
     way of setting SW?))

(DEFN PUSHF (WRD1 WRD2 ST)
 (SET.PC (B930.ADD.8BIT (PC ST) 2)
       (SET.MEM (B930.ADD.8BIT (FETCH (ACS ST) (B WRD2)) 1)
                (SW ST)
                (SET.AC (B WRD2)
                        (B930.ADD.8BIT (FETCH (ACS ST)
                                              (B WRD2))
                                       1)
                        ST)))
 (* See POPF))

(DCL EXECR (WRD1 WRD2 ST)(* We should think carefully about
                            what the PC is set to when the instr
                            is executed.))
(DEFN EXEC07 (WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 13)
     (IF (AND (EQUAL (OP1 WRD2) 1)
              (EQUAL (AC WRD2) 1))
         (LDM WRD1 WRD2 ST)
         (SET.ERROR T ST))
 (IF (EQUAL (OP2 WRD1) 14)
     (IF (AND (EQUAL (OP1 WRD2) 1)
              (EQUAL (AC WRD2) 1))
         (STM WRD1 WRD2 ST)
         (SET.ERROR T ST))
 (IF (EQUAL (OP2 WRD1) 15)
     (IF (AND (EQUAL (A WRD1) 0)
              (EQUAL (IBIT WRD2) 0)
              (EQUAL (OP1 WRD2) 4))
         (PADDM/POPM/PUSHM WRD1 WRD2 ST)
         (SET.ERROR T ST))
     (SET.ERROR T ST))))
 (* The ISP says that any OP2 other than 13, 14, and 15 is a no op;
    we say error.  The programmers manual implies that in addition to
    the conditions on OP2 of WRD1 there are specific bit patterns required
    in WRD2.  We cause errors if these bits are not set correctly.
    EXCEPT, the manual says that A of WRD1 in PUSHM is don't care and
    we require zeroes as in PADDM.))


(DEFN EXEC00 (WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 0)
     (TRA/NOP WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 1)
     (DECEQ WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 2)
     (LCM WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 3)
     (RLS WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 4)
     (CONT WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 5)
```

```
      (DECNE WRD1 WRD2 ST)
  (IF (EQUAL (OP2 WRD1) 6)
      (ANDOP WRD1 WRD2 ST)
  (IF (EQUAL (OP2 WRD1) 7)
      (RLL WRD1 WRD2 ST)
  (IF (EQUAL (OP2 WRD1) 8)
      (ADDR WRD1 WRD2 ST)
  (IF (EQUAL (OP2 WRD1) 9)
      (IR/CLA WRD1 WRD2 ST)
  (IF (EQUAL (OP2 WRD1) 10)
      (OROP WRD1 WRD2 ST)
  (IF (EQUAL (OP2 WRD1) 11)
      (MPY WRD1 WRD2 ST)
  (IF (EQUAL (OP2 WRD1) 12)
      (CLAO/SUBR WRD1 WRD2 ST)
  (IF (EQUAL (OP2 WRD1) 13)
      (ACM WRD1 WRD2 ST)
  (IF (EQUAL (OP2 WRD1) 14)
      (CMPR WRD1 WRD2 ST)
      (DIV WRD1 WRD2 ST)))))))))))))))

(DEFN EXECF (WRD1 WRD2 ST)
 (IF (EQUAL (OP3 WRD1)  0)
     (SFE1 WRD1 WRD2 ST)
 (IF (EQUAL (OP3 WRD1)  1)
     (SFE2 WRD1 WRD2 ST)
 (IF (EQUAL (OP3 WRD1)  2)
     (SFE3 WRD1 WRD2 ST)
 (IF (EQUAL (OP3 WRD1)  3)
     (SRIE WRD1 WRD2 ST)
 (IF (EQUAL (OP3 WRD1) 4)
     (SROV WRD1 WRD2 ST)
 (IF (EQUAL (OP3 WRD1) 5)
     (SFIR WRD1 WRD2 ST)
 (IF (EQUAL (OP3 WRD1) 6)
     (SRF1 WRD1 WRD2 ST)
 (IF (EQUAL (OP3 WRD1) 7)
     (SRF2 WRD1 WRD2 ST)
 (IF (EQUAL (OP3 WRD1) 8)
     (STE1 WRD1 WRD2 ST)
 (IF (EQUAL (OP3 WRD1) 9)
     (STE2 WRD1 WRD2 ST)
 (IF (EQUAL (OP3 WRD1) 10)
     (STE3 WRD1 WRD2 ST)
 (IF (EQUAL (OP3 WRD1) 11)
     (SSIE WRD1 WRD2 ST)
 (IF (EQUAL (OP3 WRD1) 12)
     (SSOV WRD1 WRD2 ST)
 (IF (EQUAL (OP3 WRD1) 13)
     (STIR WRD1 WRD2 ST)
 (IF (EQUAL (OP3 WRD1) 14)
     (SSF1 WRD1 WRD2 ST)
     (SSF2 WRD1 WRD2 ST)))))))))))))))

(DEFN EXEC10 (WRD1 WRD2 ST)
  (IF (EQUAL (OP2 WRD1) 0)
      (SLSA WRD1 WRD2 ST)
  (IF (EQUAL (OP2 WRD1) 1)
      (SLLA WRD1 WRD2 ST)
```

```
(IF (EQUAL (OP2 WRD1) 2)
    (SKGT WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 3)
    (SKLT WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 4)
    (SLSL WRD1 WRD2 ST)
(IF (EQUAL (OP2 WRD1) 5)
    (SLLL WRD1 WRD2 ST)
(IF (EQUAL (OP2 WRD1) 6)
    (SKGE WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 7)
    (SKLE WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 8)
     (SRSA WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 9)
     (SRLA WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 10)
     (SKEQ WRD1 WRD2 ST)
(IF (EQUAL (OP2 WRD1) 11)
     (EXECF WRD1 WRD2 ST)
(IF (EQUAL (OP2 WRD1) 12)
    (SRSL WRD1 WRD2 ST)
(IF (EQUAL (OP2 WRD1) 13)
    (SRLL WRD1 WRD2 ST)
(IF (EQUAL (OP2 WRD1) 14)
    (SKNE WRD1 WRD2 ST)
    (IAR WRD1 WRD2 ST))))))))))))))))

(DEFN EXEC17 (WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 0)
    (DADDR WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 1)
    (DSUBR WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 3)
    (EXOR WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 12)
    (HALT WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 13)
    (RET WRD1 WRD2 ST)
 (IF (EQUAL (OP2 WRD1) 15)
    (IF (EQUAL (OP1 WRD2) 1)
        (IF (EQUAL (AC WRD2) 0)
            (JSS WRD1 WRD2 ST)
        (IF (EQUAL (OP2 WRD2) 2)
            (IF (EQUAL (IBIT WRD2) 0)
                (RPS WRD1 WRD2 ST)
                (SET.ERROR T ST))
        (IF (EQUAL (OP2 WRD2) 14)
            (IF (EQUAL (IBIT WRD2) 0)
                (EXECR WRD1 WRD2 ST)
                (SET.ERROR T ST))
            (SET.ERROR T ST))))
   (IF (EQUAL (OP1 WRD2) 2)
       (IF (EQUAL (OP2 WRD2) 0)
           (IF (EQUAL (IBIT WRD2) 0)
               (DMPY WRD1 WRD2 ST)
               (SET.ERROR T ST))
       (IF (EQUAL (OP2 WRD2) 1)
           (IF (EQUAL (IBIT WRD2) 0)
```

```
                (DACM WRD1 WRD2 ST)
                (SET.ERROR T ST))
            (SET.ERROR T ST)))
    (IF (EQUAL (OP1 WRD2) 4)
        (IF (EQUAL (OP2 WRD2) 5)
            (IF (EQUAL (IBIT WRD2) 0)
                (POPF WRD1 WRD2 ST)
                (SET.ERROR T ST))
            (IF (EQUAL (OP2 WRD2) 6)
                (IF (AND (EQUAL (IBIT WRD2) 0)
                         (EQUAL (A WRD2) 15))
                    (PUSHF WRD1 WRD2 ST)
                    (SET.ERROR T ST))
                (SET.ERROR T ST)))
        (SET.ERROR T ST))))
     (SET.ERROR T ST)))))))

(DEFN EXECUTE (WRD1 WRD2 ST)
 (IF (EQUAL (OP1 WRD1) 0)
  (IF (EQUAL (IBIT WRD1) 0)
      (EXEC00 WRD1 WRD2 ST)
      (EXEC10 WRD1 WRD2 ST))
 (IF (EQUAL (OP1 WRD1) 1)
     (IF (EQUAL (AC WRD1) 0)
         (JU WRD1 WRD2 ST)
     (IF (EQUAL (AC WRD1) 1)
         (JSA0 WRD1 WRD2 ST)
     (IF (EQUAL (AC WRD1) 2)
         (JSA1 WRD1 WRD2 ST)
         (JMA0 WRD1 WRD2 ST))))
 (IF (EQUAL (OP1 WRD1) 2)
     (ADD WRD1 WRD2 ST)
 (IF (EQUAL (OP1 WRD1) 3)
     (SUB WRD1 WRD2 ST)
 (IF (EQUAL (OP1 WRD1) 4)
     (CMP WRD1 WRD2 ST)
 (IF (EQUAL (OP1 WRD1) 5)
     (LOAD WRD1 WRD2 ST)
 (IF (EQUAL (OP1 WRD1) 6)
     (STO WRD1 WRD2 ST)
     (IF (EQUAL (IBIT WRD1) 0)
         (EXEC07 WRD1 WRD2 ST)
         (EXEC17 WRD1 WRD2 ST)))))))))

(DEFN B930 (ST INSTR.CNT INTER.INSTR.LST)
  (IF (ZEROP INSTR.CNT)
      (LIST ST INSTR.CNT)
      (IF (ERROR ST) (LIST ST INSTR.CNT)
          (IF (HALT ST) (LIST ST INSTR.CNT)
              (B930
               (IF (AND (IE ST)
                        (MEMBER INSTR.CNT INTER.INSTR.LST))
                   (EXECUTE (FETCH (MEM ST) 2001Q) (FETCH (MEM ST) 2002Q) ST)
                   (EXECUTE (FETCH (MEM ST) (PC ST))
                            (FETCH (MEM ST) (PC+1 ST))
                            ST))
               (SUB1 INSTR.CNT)
               INTER.INSTR.LST))))
  (* We assume from Chuck's code rather than the programmers manual
```

or ISP that interrupts jump to 2001 octal = 2001Q
For all purposes in the execution of the instruction at 2001,
the PC points to the instruction we were about to execute.)

)))

STOP