The SIFT Hardware/Software Systems - Volume II
Software Listings

Daniel L. Palumbo

September 1985

Date for general release September 30, 1987
Introduction ................................................................. 1
MODULE SIFTDEC.CON .................................................. 2
MODULE SIFTDEC.TYP .................................................. 4
MODULE SIFTDEC.GLO .................................................. 5
MODULE SIFTOP.MCP .................................................. 7
*** GPROCESSOR ....................................................... 7
*** DBADDR ............................................................. 8
*** BROADCAST ......................................................... 8
*** STOBROADCAST .................................................... 9
*** WAITBROADCAST ................................................... 9
*** WORK ............................................................... 10
*** SINCTH ............................................................. 10
*** FAIL ............................................................... 10
*** ERR ............................................................... 10
*** VOTE5 ............................................................. 11
*** VOTE3 ............................................................. 12
*** VOTE ............................................................. 13
*** GETVOTE .......................................................... 14
*** VSCHEDULE ....................................................... 14
*** TSCHEDULE ....................................................... 15
*** BUILDTASK ....................................................... 15
*** SCHEDULER ....................................................... 16
*** NULLTASK ......................................................... 16
*** ERRTASK ........................................................ 17
*** FAULTISOLATIONTASK ........................................... 18
*** CLRBUFS .......................................................... 19
*** RECBUF S .......................................................... 19
*** XPREF ............................................................. 20
*** RECF TASK ........................................................ 21
*** CLKTASK .......................................................... 22
*** INITIALIZE ........................................................ 24
MODULE SIFTIC.MCP .................................................. 26
*** ICT1 ............................................................... 27
*** RANDOMIZE ....................................................... 27
*** COMUN1553A ...................................................... 27
*** WAIT1553A ....................................................... 27
*** GETNDR ........................................................... 28
*** GETREALDATA .................................................... 29
*** PROCEDURE GETRANDOMDATA ................................ 29
*** PROCEDURE GETNEWDATA ...................................... 30
*** ICT2 ............................................................... 32
*** REBROADCAST .................................................... 32
*** ICT3 ............................................................... 34
*** GETIC2PROC ...................................................... 34
*** VOTEDATA ........................................................ 35
*** RESTORE .......................................................... 36
*** MEDIAN ............................................................ 37
*** ICINIT ............................................................. 38
MODULE SIFTIH.SR .................................................... 39
MODULE SCHEDULE.SR ................................................ 44
MODULE GLOBALS.SR ................................................ 55
MODULE SIFTAP.MCP .................................................. 57
*** ICOS ............................................................... 58
*** ISIN ............................................................... 58
*** ISQRT ............................................................. 58
27-JUN-85 The SIFT Hardware/Software Systems - Volume II
Software Listings

*** MLS ................................................................. 59
*** GUIDANCE ............................................................. 60
*** LATERAL ............................................................... 62
*** PITCH ................................................................. 63
*** APPINIT ............................................................... 64
MODULE APPLMD.SR .................................................... 65
Introduction

This document contains software listings of the SIFT operating system and application software. The software is coded for the most part in a variant of the Pascal language, Pascal*. Pascal* is a cross-compiler running on the VAX and Eclipse computers. The output of Pascal* is BDX-390 assembler code. When necessary, modules were written directly in BDX-930 assembler code. The listings in this document supplement the description of the SIFT system found in Volume I of this report, "A Detailed Description".
27-JUN-85 The SIFT Hardware/Software Systems - Volume II
Software Listings

MODULE SIFTDEC.COM

const
maxprocessors = 8; (* highest processor number *)
tasks = 12; (* number of tasks in the system *)
maxframe = 7; (* Maximum frames in a cycle *)
maxsubframe = 26; (* last subframe in a frame *)
maxsched = 6; (* highest schedule configuration *)
maxdata = 1015; (* highest address in the datafile *)
maxtrans = 1023; (* highest address in the trans. file *)
maxdb = 127; (* highest address in a databuffer *)
dbsize = 128; (* size of a databuffer *)
maxbink = 200; (* maximum size of buffer information table *)
maxbufs = 119; (* maximum number of buffers. *)
maxstate = 128; (* largest number of items in a statevector *)
tentrysize = 5*maxstate; (* size of a task entry *)
ttsize = tentrysize*(tasks+1); (* size of the task table. *)
maxreconfig = 16#6FF; (* maximum size of schedule table (1791) *)
base = 896; (* minimum value of the transaction pointer *)
eofbit = 16#8000; (* end of file bit for transaction *)
max_window = 160; (* length of window in clock task (250)*)

(* the following are constants to be used when referring to buffers. *)
(* reserved buffers *)
. . .

(* unused buffers *)
. . .

(* system buffers *)
errerr=33;
gexecrecon=34;
gexecmemory=35;
extected=36;
lock=37;
ndr=38;
xreset=39;

(* redundant 1553a data is input into a, b or c buffers
for p's 1, 2 and 3 respectively *)
astart=40; (* must correspond to first of a series *)
aalpha=40; abeta=41; acmdalt=42; acmdhead=43; adistance=44;
aglideslope=45; alocalizer=46; ap=47; aphi=48; apan=49;
apsi=50; aq=51; ar=52; aradius=53; arturn=54; atheta=55;
au=56; ax3=57; axcnt=58; ay=59; ay0=60;
alast=60; (* must correspond to last of a series *)
balpha=61; bbeta=62; bcmmdalt=63; bcmmdhead=64; bdistance=65; 
bglideslope=66; blocalizer=67; bp=68; bphi=69; bphitrn=70; 
bpri=71; bq=72; br=73; bradius=74; bturn=75; btheta=76; 
bu=77; bx3=78; bxctr=79; by3=80; byctr=81; 
calphi=82; cbeta=83; ccmdalt=84; ccmdhead=85; cdistance=86; 
cglideslope=87; clocalizer=88; cp=89; cphi=90; cphitrn=91; 
cpsi=92; cq=93; cr=94; cradius=95; cturn=96; ctheta=97; 
cu=98; cx3=99; cxctr=100; cy3=101; cyctr=102; 
(* The o series are the 1553a output values. *)

ostart=103; (* must correspond to first of o series *)
ocmdall=103; ocmddele=104; ocmdrud=105; ocmdthr=106; 
odely=107; odelz=108; opitmo=109; olatmo=110; oreconf=111; 
olast=111; (* must correspond to last of o series *)
osynch=112;

(* Internal values. *)

phin=113; psrn=114; rn=115; 
qx=116; qy=117; qz=118; timer=119;

(* end of buffer definitions *)

(* 1553a constants *)

appnum = timer-ostart+1; (* number of 1553 broadcast buffers *)
onum = ostart; (* beginning of saved region *)
num!553a=alast-astart+1; (* number of items to read *)
onum!553a=olast-ostart+1; (* number of items to write *)
bas1553a=tpbase+astart; (* first input location *)
mas1553a=16#00FF; (* status bits *)
out1553a=olast-ostart+1; (* number of items to transmit *)
obas1553a=tpbase+ostart; (* first output location. *)
sa=0; (* subaddress 0 *)
sa1=16#20; (* subaddress 1 *)
rec1553a=16#00; (* Receive *)
tra1553a=0; (* Transmit *)
rtl=16#800; (* remote terminal 1 *)
sbas1553a=tpbase+osynch; (* synch word. *)

(* the following constants are to be used when refering to task_ids. *)

zerot=0; (* the zero task *)
nullt=1; (* the null task *)
clktid=2; (* the clock task *)
clktid=3; (* ic task 1 *)
ictid=4; (* ic task 2 *)
ictid=5; (* ic task 3 *)
errtid=6; (* the error task *)
fitsid=7; (* the fault isolation task *)
rcftid=8; (* the reconfiguration task *)
MODULE SIFTDEC.TYP

type

dfindex=0..maxdata;
  (* data file *)
dftype=array[dfindex] of integer;
  (* transaction file *)
  tpindex=0..maxtrans;
  (* transaction file *)
tftype=array[tpindex] of integer;
  processor=1..maxprocessors; (* processor *)
  procint=array[processor] of integer;
  probool=array[processor] of boolean;
  buffer=0..maxbufs; (* one for each buffer. *)
  bufint=array[buffer] of integer;
  bufrec=record
    dbx:integer;
    ad:procint;
    end;
  statevector=array[0..maxstate] of integer;
  sched_call=(tasktermination,clockinterrupt,systemstartup);
  taskentry=record
    status:sched_call; (* cause of the last pause. *)
    bufs:integer; (* ptr to list of bufs broadcasted. *)
    errors:integer; (* Number of task overrun errors. *)
    stkptr:integer; (* last stack pointer *)
    state:statevector; (* stack for task *)
    end;
  task=0..tasks; (* one for each task. *)
  dbindex=0..maxdb; (* data buffer *)
  bitmap=0..255; (* vector of bits 0..7 *)
  schindex=0..maxreconfig; (* schedule table index *)
27-JUN-85 The SIFT Hardware/Software Systems - Volume II
Software Listings

MODULE SIFTDEC.GLO

(* the following constants specify the absolute addresses of the fixed
data structures. Some data structures are fixed due to hardware
constraints. Others are global variables, and fixing their address
is the only way to reference them globally. *)

(* note siftdec.glo supplies the global symbols to Pascal modules. File
globals.ar supplies the linker with symbol names for these locations.
Both files should be maintained *)

const
  tfloc=16#3400;           (* Address of transaction file. *)
  gfrloc=16#3800;          (* Address of global frame count *)
  sfloc=16#3801;           (* Address of subframe count *)
  dbloc=16#3802;           (* Address of dbad. *)
  rploc=16#3810;           (* Address of rpont *)
  stackloc=16#5000;        (* "Exec Stack" location - sifth *)
  tloc=16#5500;            (* Address of tt. *)
  bloc=16#6000;            (* Address of bt. *)
  numloc=16#6800;          (* Address of numworking. *)
  pidloc=16#6801;          (* Address of pid. *)
  vtorloc=16#6802;         (* Address of vtor. *)
  rtovloc=16#680A;         (* Address of rtov. *)
  pvloc=16#6840;           (* Address of post vote buffer. *)
  sloc=16#6D00;            (* Address of scheds. *)
  dfloc=16#7400;           (* Address of datafile. *)
  pfloc=16#77F8;           (* Address of pideof. *)
  tploc=16#77F9;           (* Address of trans pointer. *)
  s15loc=16#77F9;          (* Address of sta1553a. *)
  clkloc=16#77FB;          (* Address of real time clock. *)
  c15loc=16#77FD;          (* Address of cmd1553a. *)
  a15loc=16#77FF;          (* Address of adr1553a. *)
  iloc=16#7800;            (* Address of buffer info. *)

var         (* the fixed address variables *)

(* pre-initialized tables.*)

  tt at tloc: array[task] of taskentry; (* Task Table *)
  scheds at sloc: array[schindex] of task; (* schedules *)
  binf at iloc: array[0..maxbinf] of buffer; (* list of tasks' buffers *)

(* hardware constrained variables *)

  transfile at tfloc: tftype;
  datafile at dfloc: dftype;
  pideof at pfloc: integer; (* processor ID discrete (read) *)
  transptr at tploc: integer; (* transaction pointer *)
  sta1553a at s15loc: integer; (* 1553a status register *)
  clock at clkloc: integer; (* real time clock (read/write) *)
cmd1553a at c15loc: integer; (* 1553a command register *)
adrl553a at a15loc: integer; (* 1553a address register *)

(* global variables *)
gframe at gfrloc: integer; (* global frame count *)
sfcount at sfcoloc: integer; (* sub frame count *)
rpcnt at rploc: integer; (* subframe repeat counter *)
postvote at pvloc: bufint; (* post vote buffer *)
dbad at dbloc: procint; (* index to start of data buffer *)
bt at bloc: array[processor,task] of bitmap; (* task bit map *)
pid at pidloc: processor; (* My processor number *)
numworking at numloc: processor; (* Number of working processors 1..8 *)
vtor at vtorloc: array[processor] of processor; (* Virtual to real processor numbers *)
rtov at rtovloc: array[processor] of processor; (* Real to virtual processor numbers *)
MODULE SIFTOP.MCP

PROGRAM SIFTOPERATINGSYSTEM;

include 'siftdec.con';
include 'siftdec.typ';
include 'siftdec.glo';

var
  working: procbool;
  errors: procint;
  v1,v2,v3,v4,v5: integer;
  p1,p2,p3,p4,p5: processor;
  taskid: task;
  presentconfig: bitmap;
  tp, vp, tpi, vpi: schindex;
  framecount: integer;
  pclock,cclock,aclock: integer;
  skew: procint;
  delta: integer;
  window: integer;
  power2: array[processor] of bitmap;
  vttodf: array[processor] of dfindex;
  nu: processor;

(* Working processors *)
(* voting *)
(* more voting *)
(* still more voting *)
(* Number of currently running task *)
(* The present configuration *)
(* schedule pointers(i.e. task, vote *)
(* start of schedule pointers *)
(* The current frame count *)
(* globals for clock synchronization *)
(* array for clock synchronization *)
(* correction applied to clock *)
(* For timing the window in clktask *)
(* power2[p] := 2**p *)
(* virtual processor to datafile address *)
(* number working processors 1..8 *)

PROCEDURE REINIT(VAR S:SCHINDEX; VAR V:STATEVECTOR); EXTERN;
PROCEDURE ICINIT; EXTERN; (* initialize interactive consistency tasks *)
PROCEDURE APPINIT; EXTERN; (* initialize applications task *)
PROCEDURE PAUSE(I:INTEGER); EXTERN; (* halt with i in R1 *)
PROCEDURE WAIT(X:INTEGER); EXTERN; (* wait x seconds *)

********** GPROCESSOR **********

PROCEDURE GPROCESSOR;
(* Set the processor pid as a number between 1 and maxprocessor. *)

begin
  pid := ((pideof div 4000B) band 16#0F);
end; (* GPROCESSOR *)
(********** DBADDRS **********)

PROCEDURE DBADDRS;
(* calculate the index of the start of each of the databuffers. *)
var
  p: processor;
  ad: dfindex;
begin
  ad := 0;
  for p := 1 to pid-1 do
    begin
      dbad[p] := ad;
      ad := ad+dbsize; (* = 128 *)
    end;
  for p := pid+1 to maxprocessor do
    begin
      dbad[p] := ad;
      ad := ad+dbsize;
    end;
  dbad[pid] := ad; (* this processors output area *)
end; (* DBADDRS *)

(********** BROADCAST **********)

GLOBAL PROCEDURE BROADCAST(B:BUFFER);
(* Broadcast buffer b. This is provided for applications tasks, and those executive tasks that don't do it themselves. *)
var
dbx,tp: dfindex;
begin
  dbx := b; tp := dbx+tpbase;
  while pideof < 0 do;
    transfile[2*tp-1023] := eofbit bor dbx*8;
    transptr := tp; (* initiate the broadcast. *)
end; (* BROADCAST *)
global procedure stobroadcast(b: buffer; v: integer); (* Store v in buffer b and broadcast it. *)

var
  dbx: buffer;
  tp: dfindex;

begin
  dbx := b; tp := dbx+tpbase; datafile[tp] := v;
  while pideof<0 do;
    transfile[2*tp-1023] := eofbit bor dbx*8;
    transptr := tp; (* initiate the broadcast. *)
  end; (* STOBROADCAST *)
end;

GLOBAL PROCEDURE WAITBROADCAST;
(* Wait for a broadcast operation to complete. *)

begin
  while pideof<0 do;
  end; (* WAITBROADCAST *)

PROCEDURE WORK;
(* At startup, identify which processors are nominally working. *)

var
  p: processor;

begin
  (* set buffer r_0 to -1 for all procs *)
  for p := maxprocessors downto 1 do datafile[dbad[p]] := -1;
  wait(1);

  (* send my pid *)
  stobroadcast(r_0,pid);
  wait(1);

  (* now see who's there *)
  for p := maxprocessors downto 1 do
    if datafile[dbad[p]] = p then
      working[p] := true
    else working[p] := false;
  working[pid] := true; (* I'm working *)

end; (* WORK *)
GLOBAL PROCEDURE SYNCH;
(* At startup synchronize the processors. Highest number processor sends
start signal *)

const
value = 16#F000;

var
p: processor;
j: dfindex;

begin
p := maxprocessors;
while not working[p] do p := p-1;

(* i points to the highest working processor. *)
j := dbad[p];
datafile[j] := 0;
if p = pid then
begin
wait(1); (* wait a second *)
stobroadcast(r_0,value); (* send signal *)
waitbroadcast; (* wait for completion *)
end
else while datafile[j]<>value do; (* wait for signal *)
end; (* SYNCH *)

PROCEDURE FAIL;
(* All returned values are wrong, so report all processors involved.
This could be coded inline, but it would take too much room. The
minor additional time that it takes to call the subroutine is
probably worthwhile. Especially since we'll probably never use it! *)

begin
errors[p1] := errors[p1]+1;
end; (* FAIL *)

PROCEDURE ERR(P: PROCESSOR);
(* Record an error for processor p. *)

begin
end; (* ERR *)
(********** VOTE5 **********)

FUNCTION VOTE5(DEFAULT:INTEGER): INTEGER;
(* This is the five way voter. Default is returned in the
 case that there is no majority value. *)

begin
  if v1 = v2 then
    if v1 = v3 then
      begin vote5 := v1;
        if v1 <> v4 then err(p4);
        if v1 <> v5 then err(p5);
        end
    else
      if v2 = v4 then
        begin vote5 := v1; err(p3);
          if v1 <> v5 then err(p5);
          end
      else
        if v1 = v5 then
          begin vote5 := v1; err(p3); err(p4); end
      end
  else
    if v3 = v4 then
      if v3 = v5 then
        begin vote5 := v3; err(p1); err(p2); end
      else
        begin vote5 := default; fail; end
    else
      begin vote5 := default; fail; end
  end

end

Page 11
else
  if v4 = v5 then
    if v2 = v4 then
      begin vote5 := v2; err(p1);
      if v2 <> v3 then err(p3);
      end
    else
      if v1 = v5 then
        begin vote5 := v1; err(p2); err(p3); end
      else
        if v3 = v5 then
          begin vote5 := v3; err(p1); err(p2); end
        else
          begin vote5 := default; fail; end
      end
  else
    if v2 = v5 then
      if v2 = v3 then
        begin vote5 := v2; err(p1); err(p4); end
      else
        begin vote5 := default; fail; end
    else
      if v2 = v3 then
        if v2 = v4 then
          begin vote5 := v2; err(p1); err(p5); end
        else
          begin vote5 := default; fail; end
        else
          begin vote5 := default; fail; end;  begin vote5 := default; err(p1); err(p2); err(p3); end;

(* VOTE5 *)

FUNCTION VOTE3(DEFAULT: INTEGER): INTEGER;
(* This is the 3 way voter. It assumes that V1 .. V3 contains
the 3 values to be voted, and that P1 .. P3 contains the
processor numbers. *)
begin
  if v1 = v2 then
    begin vote3 := v1;
    if v1<>v3 then err(p3);
    end
  else
    if v1 = v3 then
      begin vote3 := v1; err(p2); end
    else
      if v2 = v3 then
        begin vote3 := v2; err(p1); end
      else
        begin vote3 := default; err(p1); err(p2); err(p3); end;
end; (* VOTE3 *)
PROCEDURE VOTE(TK: TASK; DEFAULT: INTEGER);
(* vote task tk. Get task processor bitmap (set P1..P5). Then vote all
task's buffers. This involves either five way or three way voting. *)

var
  i,j,preal: processor;
  k: bitmap;
  b: buffer;
  d1,d2,d3,d4,d5: dfindex;
  lbufs: integer;

begin
  j := 0; i := 1;
  k := bt[nw,tk];
  repeat
    if odd(k) then (* then proc i produced task tk *)
      begin
        j := j+1;
        preal := vtor[i]; (* use real numbers for errors array access *)
        case j of
          1:begin P1:=preal; D1:=vtodf[i]; end;
          2:begin P2:=preal; D2:=vtodf[i]; end;
          3:begin P3:=preal; D3:=vtodf[i]; end;
          4:begin P4:=preal; D4:=vtodf[i]; end;
          5:begin P5:=preal; D5:=vtodf[i]; end;
          end;
        k := k div 2;
        i := i+1;
      until i > maxprocessors;
  lbufs := tt[tk].bufs;
  b := binf[lbufs]; (* first buffer *)
  if j < 3 then (* no vote *)
    while b>0 do
      if j>0 then (* use P1's value *)
      begin
        postvote[b]:= datafile[D1 + b];
        datafile [tpbase + b]:= postvote[b];
        lbufs:=lbufs+1;
        b:=binf[lbufs]; (* next buffer *)
      end
    else
      begin
        postvote[b]:= default;
        datafile [tpbase + b]:= postvote[b];
        lbufs:=lbufs+1;
        b:=binf[lbufs]; (* next buffer *)
      end;
end;
else  
  if j<5 then  
    while b>0 do  
      begin  
        V1 := datafile[D1+b];  
        V2 := datafile[D2+b];  
        V3 := datafile[D3+b];  
        postvote[b] := vote3(default);  
        datafile[tpbase+b] := postvote[b];  
        lbufs := lbufs+1;  
        b := binfo[lbufs]; (* next buffer *)  
      end  
    end  
  else  
    while b>0 do  
      begin  
        V1 := datafile[D1+b];  
        V2 := datafile[D2+b];  
        V3 := datafile[D3+b];  
        V4 := datafile[D4+b];  
        V5 := datafile[D5+b];  
        postvote[b] := vote5(default);  
        datafile[tpbase+b] := postvote[b];  
        lbufs := lbufs+1;  
        b := binfo[lbufs]; (* next buffer *)  
      end; (* VOTE *)

(*GETVOTE*)

GLOBAL FUNCTION GETVOTE(B: BUFFER): INTEGER;  
(* the getvote function is how application task access the postvote  
array. this way they arent mapped to the postvote area. *)

begin  
  getvote := postvote[b];
end; (* GETVOTE *)

(*VSCHEDULE*)

PROCEDURE VSCHEDULE;  
(* Vote those items scheduled for this subframe. *)  

var  
  tk: task;

begin  
  tk := scheds[vp]; (* get taskid to vote *)  
  while tk>0 do  
    begin  
      vote(tk,-1);  (* default = -1 *)  
      vp := vp+1;  
      tk := scheds[vp] (* get next taskid *)  
    end; (* while *)
if tk >= 0 then vp := vp+1;(* tk=-1 is end of schedule *)
end; (* VSCHEDULE *)

PROCEDURE TSCHEDULE;
(* Find the next task to schedule. *)
var
  tk: task;
begin
  tk := scheds[tp];
  if tk = -1 then begin (* end of schedule *)
    begin
      taskid := nullt; (* default to null task *)
      rpont := -2; (* 2 ticks 3.2ms *)
    end
  else begin
    begin
      taskid := tk; (* set up taskid *)
      tp := tp + 1;
      rpont := -scheds[tp]; (* load interrupt repeat counter *)
      tp := tp + 1;
    end;
  end; (* TSCHEDULE *)

PROCEDURE BUILDTASK(TASKNAME: TASK);
(* Initialize a task table entry *)
begin
  reinit(tt[TASKNAME].stkptr,tt[TASKNAME].state);
  tt[TASKNAME].status := tasktermination;
end; (* buildtask *)
GLOBAL FUNCTION SCHEDULER(CAUSE: SCHED_CALL; STATE: INTEGER): INTEGER;
(* save task stack pointer, if clock Interrupt and not nullt task
and not zero task (system startup) and not suspendable then rebuild
task. then get new subframe, next task, do vote. if task termination
select nullt task. return new task stack pointer. *)

begin
  tt[taskid].stkptr := state;
  if cause<>tasktermination then (* --- clock interrupt --- *)
    begin
      if (taskid<>nullt) then (* nullt can be interrupted *)
        if taskid<>0 then (* zero task is at system startup *)
          begin
            tt[taskid].errors := tt[taskid].errors+1;
            pause(16#BAD0 bor taskid);
            buildtask(taskid);
            end
          else tt[taskid].status := clockinterrupt;
    begin
      if sffcount >= maxsubframe then (* new frame *)
        begin
          if framecount >= maxframe then framecount := 0
          else framecount := framecount+1;
          gframe := gframe+1;
          sffcount := 0; vp := vpi; tp := tpi;
          end
        else sffcount := sffcount+1;
      tschedule;
    (* changes taskid and rpoint *)
  vschedule; (* the vote *)
  end
else (* task termination start null task *)
  taskid := nullt;
  scheduler := tt[taskid].stkptr;
end; (* SCHEDULER *)

GLOBAL FUNCTION NULLTASK: INTEGER;
(* This is the task that wastes time. It never terminates. In
the final system the nulltask will be the diagnostic task. *)

begin
  while true do (* loop forever *)
end; (* NULLTASK *)
GLOBAL FUNCTION ERRTASK: INTEGER;
(* Compute and broadcast a word with bits 7 through 0 indicating whether processors 8 through 1 have failed (1) or are ok (0). *)

const
threshold = 3;

var
err: bitmap;
i: processor;

begin
err := 0; i := maxprocessors;
repeat
err := err*2;
if (not working[i]) or (errors[i]>threshold) then err := err+1;
errors[i] := 0; (* clear error count every frame *)
i := i-1
until i < 1;
stobroadcast(errerr, err);

errtask := 0;
end; (* ERRTASK *)
GLOBAL FUNCTION FAULTISOLATIONTASK: INTEGER;
(* Compare values from the errtasks. Processors that are reported
by two or more processors (other than itself) for more than
one frame, are considered bad. The rest are considered good.
The report consists of a word, bits 7 through 0 of which
represent processors 8 through 1. (1 failed, 0 working.) *)

var
errpt: array[processor] of bitmap;
bitest,reconf: bitmap;
pi,pj: processor;
count: integer;

begin
(* load all error reports from the datafile *)
for pi := 1 to maxprocessor do errpt[pi] := datafile[dbad[pi] + errerr];

reconf := 0;
bitest := 1;
(* start with everyone working *)
(* processor 1 = bit 0, .. *)
for pi := 1 to maxprocessor do (* is pi faulty ? *)
begin
count := 0;
(* to count # of pi's accusers *)
for pj := 1 to maxprocessor do (* ask pj if pi faulty *)
if working[pj] then (* only if pj working, and *)
if pj <> pi then (* pj isn't pi ! *)
if (errpt[pj] band bitest) > 0 then (* test *)
count := count + 1; (* countem *)
if count > 1 then reconf := reconf + bitest; (* if > 1 markem bad *)
bitest := bitest*2; (* look at next pi *)
end;

(* remove processor if faulty for two consecutive frames *)
(* send resultant configuration word *)
stobroadcast(gexecreconf,reconf band postvote[gexecmemory]);
waitbroadcast;
stobroadcast(gexecmemory,reconf); (* remember this frame's result *)

faultisolationtask := 0

end; (* FAULTISOLATIONTASK *)
PROCEDURE CLRBUFS;
(* Set the buffer table so that no assumptions are made about what processor is computing the task. *)

var 
p: processor;
tk: task;

begin
   for p := 1 to maxprocessors do
      for tk := 0 to tasks do 
         bt[p,tk] := 0;
   end; (* clrbufs *)

procedure recbufs(nwk ,p: processor; s: schindex);
(* s points to the task schedule corresponding to virtual processor p. Figure out which buffers the processor will compute and mark its bit in the bt array. the voter will use the resulting bit map to figure where in the datafile to find good data to vote *)

var 
t: task;

begin
   s := s+3;
   while scheds[s]<-1 do 
      if scheds[s] = nullt then (* repeat count would follow *) 
         s := s+2
      else
         begin
            t := scheds[s];
            bt[nwk,t] := bt[nwk,t] bor power2[p];
            s := s + 2; (* next task, skip repeat count *)
         end;
   end; (* recbufs *)
FUNCTION XRECF(RECONF: BITMAP): INTEGER;
(* from reconf compute working and real to virtual map (rtov) virtual to real map (vtor), virtual to datafile offset and number working (nw). get schedule pointers according to nw. This is done even if configuration hasn't changed to insure validity of the local variables *)

var
p: processor;
s: schindex;
r: bitmap;

begin
nw := 0; p := 1; r := reconf;
repeat (* rebuild local configuration dependent data *)
  if odd(r) then (* not working *)
    begin
      working[p] := false;
      rtov[p] := maxprocessors;
    end
  else (* working *)
    begin
      working[p] := true;
      nw := nw+1;
      vtor[nw] := p;
      rtov[p] := nw;
      vtdf[nw] := dbad[p];
    end;
  r := r div 2;
p := p+1;
until p > maxprocessors;
presentconfig := reconf; (* configuration might not have changed *)
datafile[tpbase+oreconf] := reconf;
s := 0; (* find schedule for.. *)
while sched[s]<nw do s := s+ sched[s+2]; (* current number working *)
tpi := 0; p := 1;
repeat
  if vtor[p] = pid then tpi := s+3; (* and in particular, me! *)
  s := s+ sched[s+2];
p := p+1
until p > nw;
if tpi=0 then pause(16#F00B); (* i've been reconfigured out, oh well *)
s := s+3; vpi := s;
(* establish vote schedule pointer *)
umworking := nw;
(* some procedures use numworking *)
xrecf := 0;
end; (* XRECF *)
GLOBAL FUNCTION RECFTASK: INTEGER;
(* The reconfiguration task calls xrecf to do the real work. Initialization
   procedure calls xrecf also *)

begin
   recftask := xrecf(postvote[gexecreconf])
end; (* RECFTASK *)
PROCEDURE ENABLE; EXTERN; (* To enable and disable the clock *)
PROCEDURE DISABLE; EXTERN; (* interrupt *)

GLOBAL FUNCTION CLKTASK: INTEGER;
(* each working processor has a window within which he's expected to
broadcast his clock. everyone else is waiting for him. when 'seen'
they compute the skew. if they time out he's unseen. the clock is then
updated according to the mean skew. p.s., you have to use global
variables when playing with the clock or the compiler might optimize
your algorithm away *)

const
omega = 134; (* above which the skew is ignored = 209*)
commdelay = 24; (* expected communications delay = 38.4*)
clk_buf = 16#8000; (* offset 0 in datafile *)
clk_trans = 769; (* 2*tpbase-1023, trans file address for clk_buf *)

var
p: processor;
um,sum,term: integer;
x: dfindex;
epsilon: integer;

begin
disable; (* dont get interrupted during transfer *)
(* or clock correction *)
for p := maxprocessors downto 1 do datafile[dbad[p]] := 0;
transfile[clk_trans] := clk_buf; (* set transaction file *)
for p := maxprocessors downto 1 do begin
  skew[p] := 0;
  window:=clock;
  if p = pid then repeat
    if pideof>0 then begin
      datafile[tpbase]:=clock;
      transptr:=tpbase;
    end;
    until clock-window > max_window

else
begin
x:=dbad[p];
pclock := datafile[x];
repeat
cclock := datafile[x];
aclock:=clock;
if cclock <> pclock then
begin
skew[p]:= cclock + commdelay - aclock;
repeat
until clock - window > max_window;
end;
until clock-window > max_window;
end;
end;

(* Calculate the clock correction. *)
sum := 0; num := 0;
for p := 1 to maxprocessors do
begin
if working[p] then
begin
term := skew[p];
if term > omega then term := 0; (* too high *)
if term < -omega then term := 0; (* too low *)
sum := sum+term;
num := num+l;
end
end;
end;

delta := (sum div num); (* the correction is simple average *)
oclock := delta+clock;
clock := oclock;
(* Adjust the clock value. *)
enable; (* ok now *)
clktask := 0;
end; (* CLKTASK *)
GLOBAL PROCEDURE INITIALIZE;
(* initialize system state variables *)

var
  p, nwk: processor;
  s: schindex;
  r, reconf: bitmap;
  b: buffer;
  tk: task;
  i: integer;

begin

(* who am i, where are the datafile buffers, whose working, sync up *)
gprocessor; dbaddr; work; synch;

clrbufs;          (* clear the bt array *)

(* create power of 2 array *)

r := 1;
for p := 1 to maxprocessor do (* build power of 2 array *)
  begin
    power2[p] := r;
    r := r*2;
  end;

(* compute bt array for every configuration *)

s := 0;
for nwk := 1 to maxsched do
  begin
    while scheds[s] <> nwk do s := s + scheds[s+2];
    (* s := schedule for nwk *)
    for p := 1 to nwk do
      begin
        recbufs(nwk, p, s);
        (* fill bt *)
        s := s + scheds[s+2];
      end;
  end;

synch;            (* that took a long time lets resynch *)

(* set some variables *)

presentconfig := 0; reconf := 0;
gframe := 0; framecount := 0; sfcount := maxsubframe;
rport := -2; taskid := zerot; (* zero task gets clock interrupt *)
clock := 0;
(* clear postvote buffer *)

for b := 0 to maxbufs do postvote[b] := 0;

(* build task state vectors *)
for tk := 0 to tasks do
    begin buildtask(tk); tt[tk].errors := 0
    end;

(* establish initial configuration *)
for p := maxprocessors downto 1 do
    begin
        errors[p] := 0;
        reconf := reconf*2;
        if not working[p] then reconf := reconf+1
    end;

postvote[gexecmemory] := reconf; (* set the transient filter *)

i := xreof(reconf); (* reconfigure *)
appinit; (* do application initialization *)
icinit; (* and interactive consistency *)

end. (* INITIALIZE, SIFTP OPERATINGSYSTEM *)
27-JUN-85 The SIFT Hardware/Software Systems - Volume II
Software Listings

MODULE SIFTIC.MCP

PROGRAM IC;

(* This module performs the Interactive Consistency algorithm. Ict1 obtains
new data from the 1553a bus and broadcasts the data. Ict2 rebroadcasts the
data. Ict3 votes the replicates and places the results in the POSTVOTE array.
Some complications are included due to the realities of this implementation.
The 1553a data (aircraft sensor data) is computed by a simulation running on
the Eclipse 250. The Eclipse doesn't always respond in time. To keep the SIFT
in action (i.e. to avoid a waitfor loop), we save the current iteration's
POSTVOTE data, "lock" the outputs and use random data until the "new data" is
available from the Eclipse. When we have new data the POSTVOTE area is
restored and the output function is unlocked *)

include 'siftdec.con';
include 'siftdec.typ';
include 'siftdec.glo';

const
    reset = -1;

type
    replicate = 1..3;

var
    expndr,ready,oldexpected:integer; (* globals for ict1 *)
    index: dfindex;
    base: buffer;
    seed,bolock: integer;

    tempvote:array[0..appnum] of integer; (* ict3: temporary storage *)
    vp:array[replicate] of processor; (* ict3: virtual processor array *)

PROCEDURE BROADCAST(B:BUFFER); EXTERN;
PROCEDURE STOBROADCAST(B:BUFFER; V:INTEGER); EXTERN;
PROCEDURE WAITBROADCAST; EXTERN;
PROCEDURE PAUSE(I:INTEGER); EXTERN;
FUNCTION GETVOTE(Q:BUFFER):INTEGER; EXTERN;
GLOBAL FUNCTION ICT1:INTEGER;

(* When output is available (unlocked), the data is sent to aircraft. all processors participating in iclt will test for arrival of new data. If data ready, receive it. If not use randomized data and lock output.*)

FUNCTION RANDOMIZE (SEED: INTEGER): INTEGER;

begin
    randomize := (25173*seed+13849) mod 65536;
end; (* RANDOMIZE *)

PROCEDURE COMUN1553A(ADR,N,SA,MODE,RT:INTEGER);

(* N words, starting at ADR, are received from/transmitted to sub-address SA, remote-terminal RT, according to MODE *)

const errmask=16#003F; (* bits 0-5 *)
var i,cmd:integer;

PROCEDURE WAIT1553A;

begin
end; (* WAIT1553A *)

begin
    cmd:=n+sa+ mode+rt;
    adr1553a:=adr;
    cmd1553a:=cmd; (* doit *)
    wait1553;

    if errmask band sta1553a <> 0 then
        begin (* try again if needed *)
            adr1553a:=adr;
            cmd1553a:=cmd; (* requires 45 + n*20 us *)
            wait1553a;
        end
    else
        begin (* allow time for retransmit *)
            b1ock:=clock;
            i:= 28 + n*(12); (* clock tick = 1.6 us *)
            while clock-b1ock < i do;
        end
end; (* COMUN1553A *)
BEGIN (* set buffer area to negative indication *)
for i:=1 to maxprocessors do datafile[dbad[i]]:=0;

(* receive new data ready from Eclipse *)
comun1553a(sbas1553a,1,sa1,rec1553a,rt1);
val:=datafile[sbas1553a]; (* val = new data ready flag *)

(* if ndr set positive indication for me *)
if (val=exndr) or (val=reset) then datafile[tpbase]:=1;

waitbroadcast;
broadcast(r_0); (* let others know *)
clock:=clock; (* begin wait *)

(* select buffer area for data *)
(* get my virtual processor *)
P:=rtov[pid];
if p > 3 then pause(16#00C1); (* should only be three *)
case p of
  1: base := aalpha;
  2: base := balpha;
  3: base := calpha;
end;
index:=base+tpbase;

while clock-bclock < Max_window do (* wait max skew *)
end; (* GETNDR *)
PROCEDURE GETREALDATA;
(* lets all read the new data flag and then read air data *)

begin
  comun1553a(sbas1553a,1,sa1,rec1553a,rt1); (* get ndr flag *)
  if datafile[sbas1553a]=reset then (* reset mode if necessary *)
    begin
      stobroadcast(xreset,1);
      expndr:=reset;
    end
  else stobroadcast(xreset,0);
  comun1553a(index,num1553a,sa0,rec1553a,rt1); (* get air data *)
  stobroadcast(ndr,1); (* unlock outputs *)
end; (* GETREALDATA *)

PROCEDURE GETRANDOMDATA;
(* there was no new data ready, so, lets substitute random data and fly *)

var i: dfindex;

begin
  stobroadcast(xreset,0);
  expndr:=oldexpected; (* set to previous iteration *)
  seed:=gframelmaxsubframemsfcount;
  for i:= 0 to (num1553a-1) do (* substitute random data *)
    begin
      seed := randomize(seed);
      datafile[i+index] := seed;
    end;
  stobroadcast(ndr,0); (* lock the outputs *)
end; (* GETRANDOMDATA *)
PROCEDURE GETNEWDATA;
(* if at least two processors have received the new data flag
use real data, else use random data *)

var p: processor;

begin
    getndr;
    ready:=0;
    for p := 1 to numworking do (* is anybody ready?? *)
        if datafile[dbad[vctor[p]]]=1 then ready := ready +1;
        if (ready>=2) or ((numworking<2) and (datafile[tpbase]=1))
            then getrealdata
        else getrandomdata;
end; (* GETNEWDATA *)

PROCEDURE DISTRIBUTE;
(* send data, real or random, to other processors *)

const
tbase = 2*tpbase-1023;

var
    b: buffer; tp: dfindex; bend: integer;

begin
    bend := base + num1553a -1;
    for b := base to bend do
        transfile[2*b+tbase]:=b*8; (* set transaction file *)

    waitbroadcast;

    (* last buffer gets eof *)
    transfile[2*(bend) + tbase]:=eofbit bor (bend*8);
    pideof:=0; (* this enables multiple broadcasts *)
    transp: = base + tbase; (* this does it *)

    waitbroadcast;
end; (* DISTRIBUTE *)
begin (* ICT1 *)
exnplr:=getvote(expected);  (* get this iterations ndr flag *)

if getvote(lock)=0 then (* send output and ndr-first time trash *)
begin
    comun1553a(obas1553a, onum1553a, sa0, tra1553a, rt1);
datafile[bsas1553a]:=expnplr;
    comun1553a(bsas1553a, 1, sa1, tra1553a, rt1);
end;
oldexpected:=expnplr;  (* save in case not ready for next iteration *)

if expnplr < 0 then expnplr := 1 (* compute next ndr flag *)
else if expnplr = 32767 then expnplr:=1
else expnplr:=expnplr+1;

getnewdata;  (* if ndr get real data else random data *)
distribute; (* broadcast to other computers *)
stobroadcast(expected, expnplr); (* save for next time *)
ict1:=0;
end; (* ICT1 *)
GLOBAL FUNCTION ICT2: INTEGER;

(* four processors run ict2. They take the input values from ict1 and rebroadcast them *)

var more: boolean;
ic1v: bitmap;
vpx,p,icolp: processor;

(******** REBROADCAST ********)

PROCEDURE REBROADCAST( VPX, P: PROCESSOR);
(* vpx = 0,1,2 corresponds to 1553 buffers a,b,c. p identifies the processor and therefore which mailbox *)

var
b,bend: buffer;
tp,k: dfindex;

begin  (* broadcast what was received from others *)

k:=dbad[p];  (* datafile offset of p's mailbox *)
b:=aalpha+(num1553a*vpx);  (* offset within mailbox *)
bend:=b+num1553a-1;  (* end of area a,b, or c *)

while b<=bend do
begin
  tp:=b+tpbase;  (* datafile offset of my output area *)
  datafile[tp]:=datafile[k+b];  (* move data *)
  transfile[2*tp-1023]:=b*8;  (* set transaction file *)
  b:=b+1
end;

waitbroadcast;
transfile[2*tp-1023]:=eofbit bor (bend*8);  (* last buffer gets eof *)
pideof:=0;  (* this enables multiple broadcasts *)
transp1r:= tp-num1553a+1;  (* this does it *)
end; (* REBROADCAST *)
begin (* ICT2 *)

(* we need to establish which processors ran ict1 *)

(* vpx keeps track of which 1553 buffers we're dealing with: a, b, or c *)

vpx:=0;

(* iclv is the virtual processor vector for ict1 *)

iclv := bt[numworking,ict1id];

(* iclp is the virtual processor number *)

iclp := 1;

repeat
    if odd(iclv) then (* then vproc iclp produced TASK ict1 *)
        if vpx < 3 then (* we always have at least 3 ict1 tasks *)
            begin
                p:=vtor[iclp]; (* p now physical proc *)
                if p <> pid (* dont broadcast my ict1 data *)
                    then rebroadcast(vpx,p);
                vpx := vpx + 1;
            end; (* if odd *);
        iclp := iclp + 1; (* query next virtual processor *)
        iclv := iclv div 2;
    until (iclp > numworking);

ict2:=0;

end; (* ICT2 *)
GLOBAL FUNCTION ICT3:INTEGER;
(* get values replicated by icl2 and vote them *)

var db: integer; (* db=0,1,2 corresponds to 1553 buffers a,b,c *)
ictlv: bitmap; (* bitmap of processors producing ict1 *)
ictlp: processor; (* virtual processor number *)
rep: replicate;

PROCEDURE GETIC2PROC (IC1P: PROCESSOR);
(* get set of processors that rebroadcast iclp's data. set is returned
in global array vp *)

var
rep: replicate; (* will get at most 3 replicates *)
ic2v: bitmap; (* bitmap of processors that produced icl2 *)
ic2p: processor; (* virtual processor number *)

begin
rep:=1; (* begin with first replicate *)
ic2p:=1; (* assume it was produced by virtual processor 1 *)
ic2v := bt[numworking,ic2id]; (* get bitmap *)

while rep<3 do (* look for at most 3 replicates *)
begin
while not odd(ic2v) do (* if odd ic2p produced icl2 *)
begin
ic2v := ic2v div 2;
ic2p := ic2p + 1;
end;

(* ic2p would not rebroadcast data it produced with ict1. if numworking
= 3 use the data originally produced by ic2p with ict1, it will be
in correct area. If numworking < 3 will use first processor's data *)
if (ic2p <> iclp) or (numworking=3) then
begin
vp[rep] := ic2p; (* save processor number *)
rep:=rep+1 (* look for next replicate *)
end; (* if ic2p *)

ic2p := ic2p + 1;
ic2v := ic2v div 2;
end; (* while rep *)
end; (* GETIC2PROC *)
(********** VOTEDATA **********)

PROCEDURE VOTEDATA(DB: INTEGER);
(* vote the data replicates for processors specified by array vp and
 variable db.  db = 0,1,2 corresspends to 1553 buffers a,b,c *)

var
 b,base,nb: buffer;
 v1,v2,v3: integer;

begin
 base:=alpha+(num1553a*db); (* begining of buffer area *)
 for b:=0 to (num1553a-1) do
 begin
(* vote each data and put in posvote array *)

 nb:=base+b; (* nb buffer number *)

(* this next statement retrieves the replicate data from the data file. the
 statement was originally broken down into a series of statements. this
 required two more local variables. the compiler couldn't handle this.
 using a function worked, but took too long. *)

 v1 := datafile[ dbad[ vtor[vp[1]]] + nb ]; (*
 the first replicate
 the virtual number of the processor that produced it
 now a physical processor number
 start of the processor's mailbox area
 the total datafile index
 the data value *)

 v2 := datafile[ dbad[ vtor[vp[2]]] + nb ]; (* second rep. *)

 v3 := datafile[ dbad[ vtor[vp[3]]] + nb ]; (* third rep. *)
if \( v_1 = v_2 \) then postvote[nb]:=v1
else
  if \( v_1 = v_3 \) then postvote[nb]:=v1
  else
    if \( v_2 = v_3 \) then postvote[nb]:=v2
    else
      pause(16#0C3); (* what we have here is a *)
      (* failure to communicate *)
end; (* for b *)
end; (* VOTEDATA *)

(*############ RESTORE ############*)

PROCEDURE RESTORE;
(* if ndr and locked then restore temporary storage and unlock, else lock outputs *)

var i: integer;

begin
  if getvote(ndr) > 0 then (* if new data is available, and *)
    begin (* or else *)
      if getvote(lock) > 0 then (* we have been locked, then *)
        begin
          stobroadcast(lock,0); (* unlock, and *)
          for i:= 0 to (appnum-1) do (* restore temporary *)
            postvote[num+i]:=tempvote[i];
        end
      end
    else (* if data not available, and *)
      if getvote(lock) = 0 then (* we are unlocked, then *)
        begin
          stobroadcast(lock,1); (* lock outputs, and *)
          for i := 0 to (appnum-1) do (* save data *)
            tempvote[i] := postvote[num+i];
        end;
    end; (* RESTORE *)

begin (* ICT3 *)
  ic1v := bt[numworking,ic1id];
  ic1p := 1;
  for db:=0 to 2 do (* for 1553 buffers a,b,c do *)
    begin
      if numworking >= 3 then (* get set of processors which *)
        begin (* produced replicates of area db *)
          while not odd(ic1v) do (* this corresponds to the *)
            begin
              ic1v := ic1v div 2;
              ic1p := ic1p + 1;
            end;
          getic2proc(ic1p);
        end (* ICT3 *)

Page 36
else (* else use processor 1 *)
    for rep:=1 to 3 do vp[rep]:=l;
end;

votedata(db); (* vote the replicates, putting results in postvote array *)

iclp := iclp + 1;
iclv := iclv div 2;
end; (* for db *)
restore; (* if we have new data, restore temporary data storage *)

ict3:=0;
end; (* ICT3 *)

(********** MEDIAN **********)

GLOBAL FUNCTION MEDIAN(Q:BUFFER):INTEGER;
(* Find the median of the a, b, and c values and set postvote buffer q and return the value. *)

var
    res,t,v1,v2,v3: integer;

begin
    v1:=postvote[q];
    if numworking<3 then res:=v1 (* default case. *)
    else begin
        v2:=postvote[q+num1553a];
        if v1=v2 then res:=v1 (* in this game a pair wins *)
        else begin
            v3:=postvote[q+2*num1553a];
            if v1>v2 then (* make v1 < v2 *)
                begin t:=v1; v1:=v2; v2:=t end;
            if v1>v3 then (* and v1 < v3 *)
                begin t:=v1; v1:=v3; v3:=t end;
            if v2>v3 then (* and v2 < v3 *)
                begin t:=v2; v2:=v3; v3:=t end;
            res:=v2
        end;
    end;
end;

datafile[tpbase+q]:=res; postvote[q]:=res; median:=res
end; (* MEDIAN *)
global procedure icinit;
var i: integer;
begin

postvote[expected]:=0;
stobroadcast(expected,0);

postvote[lock] := 0;
stobroadcast(lock,0);

for i:= 0 to (appnum-1) do
begin
  tempvote[i] := 0;
  postvote[onum+i]:=0;
end;

postvote[olatmo]:=1;
postvote[opitmo]:=1;

end; (* ICINIT, IC *).

(* we start with 0 as expected flag *)

(* outputs unlocked *)

(* clear temporary area *)

(* or else these guys dont broadcast, oy *)
NAME ASSEM

TITLE SIFT: Interrupt handler

The Interrupt handler for the SIFT operating system handles clock
interrupts, task termination, and system startup.

There are also routines to initialize and reinitialize state vectors.
These routines save the state of the currently running task, and then
transfer control to the (pascal) scheduler who will start up
a new task after restoring its state.

Saving the state: The following is saved in order:
1. R0
2. Flags
3. R1-R13
4. PC
R14 should not be saved as it is the heap pointer. NEW should
be noninterruptible for this reason, but since SIFT doesn't use
NEW it isn't a problem. At this point we change over to the
"exec" stack which will be initialized with the function code
(termination,clocktick,startup) and the top of the task stack
which needs to be saved in the task table for the currently
running process. The index of the currently running process
is in the global variable TSKID.

ABS
ORG 100H Starting location
CONT ER,1S Disable interrupts for initialization
JU* ASIFT Go execute.
ASIFT LINK SIFT
ORG 400H Address of real time clock interrupt
HALT Halt on powerfail
JMAO* ACINT Go to the realtime routine.

ACINT is location 40H and set up by a DEFPZ
instruction to point to label CINT. The DEFPZ
is invoked after CINT to avoid an error.

RET 0 INTERRUPT 2
RET 0 INTERRUPT 3
RET 0 ONEInterrupt 4
* ORG  3400H The transaction file
    BSZ  1024
* ORG  7400H The datafile
    BSZ  1016

* Code to start up the scheduler initially.
* This code is much like the TTERM and CINT, but it is called directly
* from pascal (it is not a return from a task termination, or clock int).

    REL

    EXTRN INITI Initializing routine in SIFTOP

    AINIT LINK INITI
    STACK FIX 5000H

    SIFT LOAD 0,STACK Pick up the stack address
    TRA 15,0 Put it in the stack pointer
    CLAO 1,1
    CLAO 2,2
    CLAO 3,3
    CLAO 4,4
    CLAO 5,5
    CLAO 6,6
    CLAO 7,7
    CLAO 8,8
    CLAO 9,9
    CLAO 10,10
    CLAO 11,11
    CLAO 12,12
    CLAO 13,13
    CLAO 14,14
    JSS* AINIT Initialize the OS
    CONT ES Allow Interrupts
    STLP JU STLP And wait for one to happen.

    ENTRY DISAB Routine called from Pascal to
    CONT ER disable interrupts.
    RPS 0

    ENTRY ENABL Routine called from Pascal to
    CONT ES enable interrupts.
    RPS 0

    RPCNT LINK 3810H Subframe repeat counter. Set in Tschedule

    ACLK FIX 1 Clock tick function code
    ASTRT FIX 2 System startup function code
    AEND FIX 17 Constant, that when added to the the base of
    a statevector, points you at the end of it.
* Code to handle task termination. This basically means setting
* things up for next time and then calling the scheduler to
* process task termination. This should run disabled
*
ENTRY TTERM
ATERM LINK TTERM
*
TTERM CONT ER disallow interrupts
LOAD 0,ATERM on task termination return here
PUSHM 0,0
PUSHM 0,0 dummy r0 save
TRA 0,15 point at top of stack
LOAD 0,-2,0 get start PC in 0
PUSHF 15 save flags
PUSHM 1,13 save registers
PUSHM 0,0 save resume PC (which is the start)
CLA0 0,0 indicate a task termination
JU SCHG to the scheduler
*
* Here is the main clock interrupt handler. By the time it
* gets called, R0 has been saved on the stack and now contains
* the resume address. Increment repeat counter and goto
* scheduler if necessary (i.e. = 0).
*
EXTRN SCHED
ASCHE LINK SCHED link to scheduler
*
CINT PUSHF 15 save the flags
PUSHM 1,1 Save a work register
LOAD* 1,RPCNT Get repeat counter
IAR 1,1 inc the counter
SKNE 1,NOINT if <> 0 restore
JU DOINT else call scheduler
*
NOINT STO* 1,RPCNT save for next time
POP M 1,1 Restore the register
POPF 15 and the flags
CONT ES Allow interrupts
RET 0 And return
*
DOINT PUSHM 2,13 Save registers (14 is heap no need to save)
PUSHM 0,0 and the resume address
LOAD 0,ACLK indicate clock interrupt
SCHG TRA 1,15 save the current stack pointer
LDM 15,15,STACK point at the executive stack
PUSHM 0,1 set function code and resume stack
JSS* ASCHE call the scheduler which is a pascal function
* which returns the new task's stack pointer
TR A 15,12 this puts it in its place
POP M 0,0 restore the resume PC to R0
POP M 1,13 restore some registers.
POPF 15 and the flags
CONT ES allow interrupts
RET 0 and go resume this routine
*
DEFPZ 40H,CINT,ACINT Map ACINT to CINT thru location 40H
Code to reinitialize a state vector

The initial stack should look like:

1. Starting address of the routine (preset in task schedule)
2. Address of TTERM
3. 15 words of nothing (r0,flags,r1-r13)
4. Starting address of the routine

REINI is a procedure called as:

```pascal
procedure reinit(var stack: integer; var state: statevector);
```

Upon exit it should set stack to point at the 4th item above.

```assembly
ENTRY REINI

REINI PUSHM 0,2
TRA 0,15
LOAD 1,-4,0 starting address of statevector
LOAD 2,0,1 get starting address of routine
STO 2,17,1 set up vector
LOAD 2,ATERM start of tterm
STO 2,1,1 save it away
ADD 1,AEND point at end of statevector
STO* 1,-5,0 return the top of stack address
POPM 0,2 restore registers
RPS 0 return
```

PAGE

TITLE SIFT: Halt (debugging) routine

```pascal
procedure pause(errocode: integer);
```

```assembly
ENTRY PAUSE

PAUSE PUSHM 0,1
TRA 0,15
CONT ER disable interrupts
LOAD 1,-3,0
HALT
CONT ES enable interrupts
POPM 0,1
RPS 0
```

* * *
TITLE SIFT: Delay routine

procedure wait(X: integer);

wait for approximately X seconds before returning.

ENTRY WAIT

WAIT PUSHM 0,3 ; SAVE SOME REGISTERS
TRX 0,15 ; POINT AT THE DISPLAY
LOAD 2,-5,0 ; GET THE NUMBER OF SECONDS
LOAD 1,F10 ; ADJUST FOR TIMING
MP1 2,1 ; MULTIPLY IT OUT
SRLA 2,1 ; RESULT IN 3
OUTER LOAD 1,HFFFF
INNER DECNE 1,INNER ; INNER LOOP TAKES ABOUT .1 SECOND
DECNE 3,OUTER ; OUTER LOOP TAKES ABOUT X SECONDS
POPM 0,3
RPS 0
HFFFF FIX OFFFFH
F10 FIX 10

function to return global clock value

TITLE GCLOCK
ENTRY GCLOC

GCLOC PUSHM 0,1
TRX 0,8
ID 12,0
POPM 0,1
RPS 0
END
27-JUN-85 The SIFT Hardware/Software Systems - Volume II
Software Listings

MODULE SCHEDULE.SR

<table>
<thead>
<tr>
<th>NAME</th>
<th>TASKT</th>
</tr>
</thead>
<tbody>
<tr>
<td>TITLE</td>
<td>SIFT: Equates</td>
</tr>
<tr>
<td>DATE</td>
<td>ABS</td>
</tr>
</tbody>
</table>

* * * *
* with new improved schedule counters *
* *
SLOC EQU 6D00H
TLOC EQU 5500H
ILOC EQU 7800H
*
* Buffer names *
*
CMDAI EQU 103
CMDEL EQU 104
CMDRN EQU 105
CMDTH EQU 106
ERRER EQU 33
EXPEX EQU 36
GEMEM EQU 35
GEREC EQU 34
LOCK EQU 37
NDR EQU 38
PHIN EQU 113
PSIN EQU 114
QDELY EQU 107
QDELZ EQU 108
QLATM EQU 110
QPITM EQU 109
QX EQU 116
QY EQU 117
QZ EQU 118
RN EQU 115
TIMER EQU 119
XRESE EQU 39
### SIFT: Task Table

```plaintext
* TITLE SIFT: Task Table
*
* * EXTRN TTERM
*
* ORG TLOC
TASK MACRO 2
EXTRN %0
FIX 0
FIX %1
FIX 0
LINK *+18
LINK %0
LINK TTERM
BSZ 15
LINK %0
BSZ 111
ENDM
*
ZTASK MACRO 1
BSZ 133
ENDM
*
T0 ZTASK 0
T1 TASK NULLT,BUF1
T2 TASK CLKTA,BUF2
T3 TASK ICT1,BUF3
T4 TASK ICT2,BUF4
T5 TASK ICT3,BUF5
T6 TASK ERRTA,BUF6
T7 TASK FAULT,BUF7
T8 TASK RECFT,BUF8
T9 TASK MLS,BUF9
T10 TASK GUIDA,BUF10
T11 TASK PITCH,BUF11
T12 TASK LATER,BUF12
```

### SIFT: Buffer Information Table

```plaintext
* TITLE SIFT: Buffer Information Table
*
* * ORG ILOC
EVENT MACRO 1
FIX %0 EVENT INDICATION
ENDM
* ```
STLOC EQU *
BUF2 EQU *-STLOC
    FIX 0
BUF6 EQU *-STLOC
    FIX 0
BUF7 EQU *-STLOC
    EVENT GERE
    EVENT GEMEM
    FIX 0
BUF10 EQU *-STLOC
    EVENT PSIN
    EVENT PHIN
    EVENT RN
    EVENT QDELY
    EVENT QLATM
    EVENT TIMER
    FIX 0
BUF3 EQU *-STLOC
    EVENT EXPEX
    EVENT XRESE
    EVENT NDR
    FIX 0
BUF4 EQU *-STLOC
    FIX 0
BUF5 EQU *-STLOC
    EVENT LOCK
    FIX 0
BUF12 EQU *-STLOC
    EVENT CMDAI
    EVENT CMDRN
    FIX 0
BUF9 EQU *-STLOC
    EVENT QX
    EVENT QZ
    EVENT QY
    FIX 0
BUF1 EQU *-STLOC
    FIX 0

CLKTA
ERRTA
FAULT
GUIDA
ICT1
ICT2
ICT3
LATER
MLS
NULLT
BUF11 EQU EVENT EVENT EVENT EVENT EVENT FIX 0
BUF8 EQU EVENT FIX 0 PAGE TITLE SIFT: Schedule Table

* ORG SLOC
SFLEN MACRO 1 FIX %0 NUMBER OF 1.6 MSEC TICKS/SUBFRAME ENDM

* SFEND MACRO 0 FIX 0 END OF VOTE FRAME ENDM

* SCHED MACRO 4 FIX %0 NUMBER OF PROCESSORS FIX %1 WHICH ONE FIX 1+%3-%2 ENDM

* SEND MACRO 0 FIX -1 END OF SCHEDULE ENDM

* VCSCD EQU 99

S11 SCHED 1,1,S11,E11 EVENT 2 CLKTA SFLEN 2 EVENT 3 ICT1 SFLEN 3 EVENT 4 ICT2 SFLEN 2 EVENT 5 ICT3 SFLEN 5 EVENT 9 MLS SFLEN 2 EVENT 10 GUIDA SFLEN 2 EVENT 11 PITCH SFLEN 2 EVENT 12 LATER SFLEN 2 EVENT 6 ERRTA SFLEN 2
27-JUN-85 The SIFT Hardware/Software Systems - Volume II
Software Listings

EVENT 1    NULLT
SFLEN 2
EVENT 3    ICT1
SFLEN 3
EVENT 4    ICT2
SFLEN 2
EVENT 5    ICT3
SFLEN 5
EVENT 9    MLS
SFLEN 2
EVENT 10   GUIDA
SFLEN 2
EVENT 11   PITCH
SFLEN 2
EVENT 12   LATER
SFLEN 2
EVENT 7    FAULT
SFLEN 3
EVENT 1    NULLT
SFLEN 2
EVENT 3    ICT1
SFLEN 3
EVENT 4    ICT2
SFLEN 2
EVENT 5    ICT3
SFLEN 5
EVENT 9    MLS
SFLEN 2
EVENT 10   GUIDA
SFLEN 2
EVENT 11   PITCH
SFLEN 2
EVENT 12   LATER
SFLEN 2
EVENT 8    RECFT
SFLEN 2

S199    SCHED 1,VCSCD,S199,E199
SFEND   0
SFEND   1
EVENT 3    ICT1
SFEND   2
SFEND   3
EVENT 5    ICT3
SFEND   4
EVENT 9    MLS
SFEND   5
EVENT 10   GUIDA
SFEND   6
EVENT 11   PITCH
SFEND   7
EVENT 12   LATER
SFEND   8
EVENT 6    ERRTA
SFEND  9
SFEND  10
EVENT  3  ICT1
SFEND  11
SFEND  12
EVENT  5  ICT3
SFEND  13
EVENT  9  MLS
SFEND  14
EVENT 10  GUIDA
SFEND  15
EVENT 11  PITCH
SFEND  16
EVENT 12  LATER
SFEND  17
EVENT  7  FAULT
SFEND  18
SFEND  19
EVENT  3  ICT1
SFEND  20
SFEND  21
EVENT  5  ICT3
SFEND  22
EVENT  9  MLS
SFEND  23
EVENT 10  GUIDA
SFEND  24
EVENT 11  PITCH
SFEND  25
EVENT 12  LATER
SFEND  26
SFEND  27
SFEND
EVENT -1

E199 SEND
*

Page 49
In the interest of efficiency, the remaining schedules are represented symbolically by the following.

### SIFT Schedules for 2 Processor

<table>
<thead>
<tr>
<th>SLOT</th>
<th>TICK</th>
<th>S21 TASK</th>
<th>S22 TASK</th>
<th>VARIABLES VOTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>CLKTA</td>
<td>CLKTA</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>ICT1</td>
<td>ICT1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT1 : EXPEX XRESE NDR</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>ICT3</td>
<td>ICT3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>MLS</td>
<td>NULLT</td>
<td>ICT3 : LOCK</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>NULLT</td>
<td>GUIDA</td>
<td>MLS : QX QZ QY</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>PITCH</td>
<td>NULLT</td>
<td>GUIDA : PSIN PHIN RN QDELY QLATM TIMER</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>NULLT</td>
<td>LATER</td>
<td>PITCH : CMDDEL QDELZ CMDTH QPITM</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>ERRTA</td>
<td>ERRTA</td>
<td>NULLT CMDAI CMDRN</td>
</tr>
<tr>
<td>10</td>
<td>22</td>
<td>NULLT</td>
<td>ERRTA</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>24</td>
<td>ICT1</td>
<td>ICT1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>27</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT1 : EXPEX XRESE NDR</td>
</tr>
<tr>
<td>13</td>
<td>29</td>
<td>ICT3</td>
<td>ICT3</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>34</td>
<td>MLS</td>
<td>NULLT</td>
<td>ICT3 : LOCK</td>
</tr>
<tr>
<td>15</td>
<td>36</td>
<td>NULLT</td>
<td>GUIDA</td>
<td>MLS : QX QZ QY</td>
</tr>
<tr>
<td>16</td>
<td>38</td>
<td>PITCH</td>
<td>NULLT</td>
<td>GUIDA : PSIN PHIN RN QDELY QLATM TIMER</td>
</tr>
<tr>
<td>17</td>
<td>40</td>
<td>NULLT</td>
<td>LATER</td>
<td>PITCH : CMDDEL QDELZ CMDTH QPITM</td>
</tr>
<tr>
<td>18</td>
<td>42</td>
<td>FAULT</td>
<td>NULLT</td>
<td>NULLT CMDAI CMDRN</td>
</tr>
<tr>
<td>19</td>
<td>45</td>
<td>NULLT</td>
<td>FAULT</td>
<td>GEREQ GEMEM</td>
</tr>
<tr>
<td>20</td>
<td>47</td>
<td>ICT1</td>
<td>ICT1</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>50</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT1 : EXPEX XRESE NDR</td>
</tr>
<tr>
<td>22</td>
<td>52</td>
<td>ICT3</td>
<td>ICT3</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>57</td>
<td>MLS</td>
<td>NULLT</td>
<td>ICT3 : LOCK</td>
</tr>
<tr>
<td>24</td>
<td>59</td>
<td>NULLT</td>
<td>GUIDA</td>
<td>MLS : QX QZ QY</td>
</tr>
<tr>
<td>25</td>
<td>61</td>
<td>PITCH</td>
<td>NULLT</td>
<td>GUIDA : PSIN PHIN RN QDELY QLATM TIMER</td>
</tr>
<tr>
<td>26</td>
<td>63</td>
<td>NULLT</td>
<td>LATER</td>
<td>PITCH : CMDDEL QDELZ CMDTH QPITM</td>
</tr>
<tr>
<td>27</td>
<td>65</td>
<td>RECFT</td>
<td>RECFT</td>
<td>LATER : CMDAI CMDRN</td>
</tr>
</tbody>
</table>
### SIFT Schedules for 3 Processors

<table>
<thead>
<tr>
<th>SLOT</th>
<th>TICK</th>
<th>S31</th>
<th>S32</th>
<th>S33</th>
<th>TASK : VARIABLES VOTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>CLKTA</td>
<td>CLKTA</td>
<td>CLKTA</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>ICT1</td>
<td>ICT1</td>
<td>ICT1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT1 : EXPEX XRESE NDR</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>ICT3 : LOCK</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>MLS : QX QZ QY</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
<td>GUIDA : PSIN PHIN RN</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>PITCH : CMDL QDELZ CMDTH</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>ERRTA</td>
<td>ERRTA</td>
<td>ERRTA</td>
<td>LATER : CMDAI CMDRN</td>
</tr>
<tr>
<td>10</td>
<td>22</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>ERRTA:</td>
</tr>
<tr>
<td>11</td>
<td>24</td>
<td>ICT1</td>
<td>ICT1</td>
<td>ICT1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>27</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT1 : EXPEX XRESE NDR</td>
</tr>
<tr>
<td>13</td>
<td>29</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>34</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>ICT3 : LOCK</td>
</tr>
<tr>
<td>15</td>
<td>36</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>MLS : QX QZ QY</td>
</tr>
<tr>
<td>16</td>
<td>38</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
<td>GUIDA : PSIN PHIN RN</td>
</tr>
<tr>
<td>17</td>
<td>40</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>PITCH : CMDL QDELZ CMDTH</td>
</tr>
<tr>
<td>18</td>
<td>42</td>
<td>FAULT</td>
<td>FAULT</td>
<td>FAULT</td>
<td>LATER : CMDAI CMDRN</td>
</tr>
<tr>
<td>19</td>
<td>45</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>FAULT: GEREQ GEMEM</td>
</tr>
<tr>
<td>20</td>
<td>47</td>
<td>ICT1</td>
<td>ICT1</td>
<td>ICT1</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>50</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT1 : EXPEX XRESE NDR</td>
</tr>
<tr>
<td>22</td>
<td>52</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>57</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>ICT3 : LOCK</td>
</tr>
<tr>
<td>24</td>
<td>59</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>MLS : QX QZ QY</td>
</tr>
<tr>
<td>25</td>
<td>61</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
<td>GUIDA : PSIN PHIN RN</td>
</tr>
<tr>
<td>26</td>
<td>63</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>PITCH : CMDL QDELZ CMDTH</td>
</tr>
<tr>
<td>27</td>
<td>65</td>
<td>RECFT</td>
<td>RECFT</td>
<td>RECFT</td>
<td>LATER : CMDAI CMDRN</td>
</tr>
<tr>
<td>SLOT</td>
<td>TICK</td>
<td>S41</td>
<td>S42</td>
<td>S43</td>
<td>S44</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>CLKTA</td>
<td>CLKTA</td>
<td>CLKTA</td>
<td>CLKTA</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>ICT1</td>
<td>ICT1</td>
<td>ICT1</td>
<td>NULLT</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT2</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>MLS</td>
<td>MLS</td>
<td>NULLT</td>
<td>MLS</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>GUIDA</td>
<td>NULLT</td>
<td>GUIDA</td>
<td>GUIDA</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>NULLT</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>NULLT</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>ERRTA</td>
<td>ERRTA</td>
<td>ERRTA</td>
<td>ERRTA</td>
</tr>
<tr>
<td>10</td>
<td>22</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>ERRTA</td>
</tr>
<tr>
<td>11</td>
<td>24</td>
<td>ICT1</td>
<td>ICT1</td>
<td>ICT1</td>
<td>NULLT</td>
</tr>
<tr>
<td>12</td>
<td>27</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT2</td>
</tr>
<tr>
<td>13</td>
<td>29</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
</tr>
<tr>
<td>14</td>
<td>34</td>
<td>MLS</td>
<td>MLS</td>
<td>NULLT</td>
<td>MLS</td>
</tr>
<tr>
<td>15</td>
<td>36</td>
<td>GUIDA</td>
<td>NULLT</td>
<td>GUIDA</td>
<td>GUIDA</td>
</tr>
<tr>
<td>16</td>
<td>38</td>
<td>NULLT</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
</tr>
<tr>
<td>17</td>
<td>40</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>NULLT</td>
</tr>
<tr>
<td>18</td>
<td>42</td>
<td>FAULT</td>
<td>FAULT</td>
<td>NULLT</td>
<td>FAULT</td>
</tr>
<tr>
<td>19</td>
<td>45</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>FAULT</td>
</tr>
<tr>
<td>20</td>
<td>47</td>
<td>ICT1</td>
<td>ICT1</td>
<td>ICT1</td>
<td>NULLT</td>
</tr>
<tr>
<td>21</td>
<td>50</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT2</td>
</tr>
<tr>
<td>22</td>
<td>52</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
</tr>
<tr>
<td>23</td>
<td>57</td>
<td>MLS</td>
<td>MLS</td>
<td>NULLT</td>
<td>MLS</td>
</tr>
<tr>
<td>24</td>
<td>59</td>
<td>GUIDA</td>
<td>NULLT</td>
<td>GUIDA</td>
<td>GUIDA</td>
</tr>
<tr>
<td>25</td>
<td>61</td>
<td>NULLT</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
</tr>
<tr>
<td>26</td>
<td>63</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>NULLT</td>
</tr>
<tr>
<td>27</td>
<td>65</td>
<td>RECFT</td>
<td>RECFT</td>
<td>RECFT</td>
<td>RECFT</td>
</tr>
</tbody>
</table>
### SIFT Schedule for 5 Processors

<table>
<thead>
<tr>
<th>SLOT</th>
<th>TICK</th>
<th>S51</th>
<th>S52</th>
<th>S53</th>
<th>S54</th>
<th>S55</th>
<th>Task: Variables Voted</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>CLKTA</td>
<td>CLKTA</td>
<td>CLKTA</td>
<td>CLKTA</td>
<td>CLKTA</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>ICT1</td>
<td>ICT1</td>
<td>ICT1</td>
<td>NULLT</td>
<td>NULLT</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>ICT2</td>
<td>ICT2</td>
<td>NULLT</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT1: EXPEX XRESE NDR</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>ICT3: LOCK</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>MLS: QX QZ QY</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
<td>GUIDA: PSIN PHIN RN QDELY QLATM TIMER</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>PITCH: CMDEL QDELZ CMDTH QPITM</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>ERRTA</td>
<td>ERRTA</td>
<td>ERRTA</td>
<td>ERRTA</td>
<td>ERRTA</td>
<td>LATER: CMDAI CMDRN</td>
</tr>
<tr>
<td>10</td>
<td>22</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>ERRTA:</td>
</tr>
<tr>
<td>11</td>
<td>24</td>
<td>ICT1</td>
<td>ICT1</td>
<td>ICT1</td>
<td>NULLT</td>
<td>NULLT</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>27</td>
<td>ICT2</td>
<td>ICT2</td>
<td>NULLT</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT1: EXPEX XRESE NDR</td>
</tr>
<tr>
<td>13</td>
<td>29</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>34</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>ICT3: LOCK</td>
</tr>
<tr>
<td>15</td>
<td>36</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>MLS: QX QZ QY</td>
</tr>
<tr>
<td>16</td>
<td>38</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
<td>GUIDA: PSIN PHIN RN QDELY QLATM TIMER</td>
</tr>
<tr>
<td>17</td>
<td>40</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>PITCH: CMDEL QDELZ CMDTH QPITM</td>
</tr>
<tr>
<td>18</td>
<td>42</td>
<td>FAULT</td>
<td>FAULT</td>
<td>FAULT</td>
<td>FAULT</td>
<td>FAULT</td>
<td>LATER: CMDAI CMDRN</td>
</tr>
<tr>
<td>19</td>
<td>45</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>FAULT: GEREQ GEMEM</td>
</tr>
<tr>
<td>20</td>
<td>47</td>
<td>ICT1</td>
<td>ICT1</td>
<td>ICT1</td>
<td>NULLT</td>
<td>NULLT</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>50</td>
<td>ICT2</td>
<td>ICT2</td>
<td>NULLT</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT1: EXPEX XRESE NDR</td>
</tr>
<tr>
<td>22</td>
<td>52</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>57</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>ICT3: LOCK</td>
</tr>
<tr>
<td>24</td>
<td>59</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>MLS: QX QZ QY</td>
</tr>
<tr>
<td>25</td>
<td>61</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
<td>GUIDA: PSIN PHIN RN QDELY QLATM TIMER</td>
</tr>
<tr>
<td>26</td>
<td>63</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>PITCH: CMDEL QDELZ CMDTH QPITM</td>
</tr>
<tr>
<td>27</td>
<td>65</td>
<td>RECF T</td>
<td>RECF T</td>
<td>RECF T</td>
<td>RECF T</td>
<td>RECF T</td>
<td>LATER: CMDAI CMDRN</td>
</tr>
</tbody>
</table>

Page 53
### SIFT SCHEDULE FOR 6 PROCESSORS

<table>
<thead>
<tr>
<th>SLOT</th>
<th>TICK</th>
<th>S61</th>
<th>S62</th>
<th>S63</th>
<th>S64</th>
<th>S65</th>
<th>S66</th>
<th>TASK: VARIABLES VOTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>CLKTA</td>
<td>CLKTA</td>
<td>CLKTA</td>
<td>CLKTA</td>
<td>CLKTA</td>
<td>CLKTA</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>ICT1</td>
<td>ICT1</td>
<td>ICT1</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>ICT2</td>
<td>NULLT</td>
<td>NULLT</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>NULLT</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>NULLT</td>
<td>MLS: QX QZ QY</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
<td>NULLT</td>
<td>PITCH</td>
<td>GUIDA: PSIN PHIN RN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GUIDA: CMDEL QDELZ CMDTH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GUIDA: QDELY QLATM TIMER</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GUIDA: QPITM</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>NULLT</td>
<td>LATER</td>
<td>LATER</td>
<td>PITCH: CMDAI CMDRN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CMDAI CMDRN</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>ERRTA</td>
<td>ERRTA</td>
<td>ERRTA</td>
<td>ERRTA</td>
<td>ERRTA</td>
<td>ERRTA</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>22</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>24</td>
<td>ICT1</td>
<td>ICT1</td>
<td>ICT1</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>27</td>
<td>ICT2</td>
<td>NULLT</td>
<td>NULLT</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT2</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>29</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>34</td>
<td>NULLT</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>36</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>NULLT</td>
<td>MLS: QX QZ QY</td>
</tr>
<tr>
<td>16</td>
<td>38</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
<td>NULLT</td>
<td>PITCH</td>
<td>GUIDA: PSIN PHIN RN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GUIDA: CMDEL QDELZ CMDTH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CMDEL QDELZ CMDTH QPITM</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>40</td>
<td>LATER</td>
<td>LATER</td>
<td>LATER</td>
<td>NULLT</td>
<td>LATER</td>
<td>LATER</td>
<td>PITCH: CMDAI CMDRN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CMDAI CMDRN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>42</td>
<td>FAULT</td>
<td>FAULT</td>
<td>NULLT</td>
<td>FAULT</td>
<td>NULLT</td>
<td>FAULT</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>45</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>47</td>
<td>ICT1</td>
<td>ICT1</td>
<td>ICT1</td>
<td>NULLT</td>
<td>NULLT</td>
<td>NULLT</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>50</td>
<td>ICT2</td>
<td>NULLT</td>
<td>NULLT</td>
<td>ICT2</td>
<td>ICT2</td>
<td>ICT2</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>52</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td>ICT3</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>57</td>
<td>NULLT</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td>MLS</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>59</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>GUIDA</td>
<td>NULLT</td>
<td>MLS: QX QZ QY</td>
</tr>
<tr>
<td>25</td>
<td>61</td>
<td>PITCH</td>
<td>PITCH</td>
<td>PITCH</td>
<td>NULLT</td>
<td>PITCH</td>
<td>GUIDA: PSIN PHIN RN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GUIDA: CMDEL QDELZ CMDTH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CMDAI CMDRN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* END
MODULE GLOBALS.SR

NAME GLOBAUS

ABS

* HERE WE FIX THE LOCATIONS OF THE GLOBAL SYMBOLS. THE ONLY NEED FOR THIS IS
* TO GIVE THESE LOCATIONS PROPER SYMBOL NAMES, WHICH PASCAL* DOES NOT
*
* NOTE SIFTDEC.GLO SUPPLIES THE GLOBAL SYMBOLS TO PASCAL MODULES. FILE
* GLOBALS.SR SUPPLIES THE LINKER WITH SYMBOL NAMES FOR THESE LOCATIONS.
* BOTH FILES SHOULD BE MAINTAINED
*
* const
*  * tfloc=16#3400;
  * tranf equ 3400h
  * gfrloc=16#3800;
  * gfram equ 3800h
  * sfloc=16#3801;
  * sfcou equ 3801h
  * dbloc=16#3802;
  * dbad equ 3802h
  * rploc=16#3810;
  * Rpcnt equ 3810h
  * stackloc=16#5000;
  * stack equ 5000h
  * tloc=16#5500;
  * tt equ 5500h
  * blocl=16#6000;
  * bt equ 6000h
  * numloc=16#6800;
  * numwo equ 6800h
  * pidloc=16#6801;
  * pid equ 6801h
  * vtorloc=16#6802;
  * vtor equ 6802h
  * rtovloc=16#680a;
  * rtov equ 680ah
  * pvlloc=16#6840;
  * postv equ 6840h
  * sloc=16#6d00;
  * sched equ 6d00h
  * dfloc=16#7400;
  * dataf equ 7400h
  * pfloc=16#77f8;
  * pflc equ 77f8h
  * tploc=16#77f9;
  * tranp equ 77f9h
  * s15loc=16#77f9;
  * sta15 equ 77f9h
  * clkloc=16#77fb;

(* Address of transaction file. *)
(* Address of global frame count *)
(* Address of subframe count *)
(* Address of dbad. *)
(* Address of rpcnt *)
(* "Exec Stack" location - sifth *)
(* Address of tt. *)
(* Address of bt. *)
(* Address of numworking. *)
(* Address of pid. *)
(* Address of vtor. *)
(* Address of rtov. *)
(* Address of post vote buffer. *)
(* Address of scheds. *)
(* Address of datafile. *)
(* Address of pideof. *)
(* Address of trans pointer. *)
(* Address of sta153a. *)
(* Address of real time clock. *)
CLOCK EQU 77FBH
* cl5loc=16#77FD;
CMD15 EQU 77FDH
* al5loc=16#77FF;
ADR15 EQU 77FFH
* 1 loc=16#800;
BINF EQU 7800H
* *
END
*these fellows perform scaling operations and are found in module applmd

where md := a*b/c;
and mdii := a*b/2**ii; *)

FUNCTION MD(A,B,C:INTEGER):INTEGER; EXTERN;
FUNCTION MD14(A,B:INTEGER):INTEGER; EXTERN;
FUNCTION MD12(A,B:INTEGER):INTEGER; EXTERN;
FUNCTION MD11(A,B:INTEGER):INTEGER; EXTERN;
FUNCTION MD10(A,B:INTEGER):INTEGER; EXTERN;
FUNCTION MD9(A,B:INTEGER):INTEGER; EXTERN;
FUNCTION MD8(A,B:INTEGER):INTEGER; EXTERN;
FUNCTION MD6(A,B:INTEGER):INTEGER; EXTERN;
FUNCTION MD2(A,B:INTEGER):INTEGER; EXTERN;
FUNCTION ICOS(X:INTEGER):INTEGER;
(* isin and icos accept arguments in the range -25736 to 25736
which is p/2 * 2**14. values of isin and icos range from
-16384 to +16384, that is, 2**14 corresponds to real value 1.0
if called with an argument outside the correct range, say 30000
the functions return values of poor accuracy. *)

var i,y:integer;

begin
  if x<0 then x:=-x;
  if x>24575 then icos:=25736-x
  else
    begin
      i := 1 + x div c1024; y := v[i];
      delty := y - v[i+1]; deltx := 1024;
      tad:=x-1024*(i-1);
      while (tad>=180) or (delty>=180) do
      begin
        deltx:=deltx div C2; delty:=delty div C2;
        if tad>deltx then
          begin y:=y-delty; tad:=tad-deltx end
        end;
      icos:=y-(tad*delty) div deltx
    end;
end; (* ICOS *)

FUNCTION ISIN(X:INTEGER) :INTEGER;
begin
  if x<0 then isin:=-icos(x+25736)
  else isin:=icos(x-25736)
end; (* ISIN *)

FUNCTION ISQRT(X:INTEGER):INTEGER;
(* the isqrt function simply hands back a negative argument.
otherwise it returns the correct value for all 16-bit inputs
less than about 32500. *)

var j,guess:integer;

begin
  if x<=1 then isqrt:=x
  else
    begin
      guess:=128; j:=1;
      while j<=7 do
        begin
guess:=(guess+x div guess) div C2; j:=j+1 end;
        isqrt:=guess
    end;
end; (* ISQRT *)
GLOBAL FUNCTION MLS:INTEGER;
(* This routine converts MLS data to x,y, and z.
Localizer > 0 is fly right. Glideslope angle is always positive. *)

begin
  d:=median(adistance); d:=-d; g:=median(aglideslope);
  l:=median(alocalizer); dist:=mdl4(d,icos(g));
  stobroadcast(qx,mdl4(dist,icos(l))); 
  stobroadcast(qy,mdl11(dist,isin(l))); 
  stobroadcast(qz,mdl10(d,isin(g))); 
  mls:=0
end; (* MLS *)
GLOBAL FUNCTION GUIDANCE:INTEGER;
(* This subroutine provides lateral GUIDANCE for the aircraft.*)

const rnav=1; intcpt=2; lclzr=3;

begin
  h:=median(acmdhead); x:=getvote(qx); y:=getvote(qy);
  r:=median(aradius); p:=getvote(psin); l:=getvote(olatmo);

  if getvote(xreset)=1 then l:=rnav;

  psiapr:=h div C2; thrsho:=md14(r,16384-icos(h));
  if h>0 then thrsho:=-thrsho;

  (* Perform mode switching logic and reset turn timer clock.*)
  ttim:=getvote(timer);
  if p<0 then p:=-p;
  if (l=rnav) and (y>thrsho) then
    begin ttim:=0; l:=intcpt end;
  if (l=intcpt) and (p<82) then l:=lclzr;
  ttim:=ttim+1;

  stobroadcast(timer,ttim);

  (* Set nominal values according to mode.*)
  if l=rnav then
    begin
      stobroadcast(psin,psiapr);
      stobroadcast(phin,0);
      stobroadcast(rn,0);
      i:=psiapr*2;
      t:=md12(y-median(ay3),icos(i));
      t:=(t-md9(x-median(ax3),isin(i)))*2;
      stobroadcast(odely,t);
    end
  else if l=intcpt then
    begin
      stobroadcast(pain,psiapr + md(ttim,median(arturn),320));
      stobroadcast(phin,median(aphitrn));
      stobroadcast(rn,median(arturn));
      t:=x-median(axcntr);
      x2:=md8(t,t);
      t:=y-median(aycntr);
      y2:=md14(t,t);
      dist:=isqrt(x2+y2)*128;
      t:=(r-dist)*8;
      if psiapr>0 then t:=-t;
      stobroadcast(odely,t);
    end

Page 60
else if l=lcizr then
    begin
    stobroadcast(psin,0);
    stobroadcast(phin,0);
    stobroadcast(rn,0);
    stobroadcast(odely,y * 8)
    end;
    stobroadcast(olatmo,1);
    guidance:=0
end; (* GUIDANCE *)
GLOBAL FUNCTION LATERAL: INTEGER;
(* lateral control. First, calculate deviations from nominal. *)

begin
  dp := median(ap);
  dr := median(ar) - getvote(rn);
  dbeta := median(abeta);
  dpsi := median(apsi) - getvote(psin);
  dphi := median(aphi) - getvote(phin);

  (* dely is not modified *)

  (* calculate aileron. *)
  t := md(-98, dp, 400) + md(98, dr, 400) + md(-6, dbeta, 8);
  t := md(-130, dphi, 100) + (t div c2);
  stobroadcast(ocmdai1, md(-6, getvote(odely), 10) + md(-102, dpsi, 200) + (t div c4));

  (* Next the rudder. *)
  t := md(8, dr, 10) + md(126, dp, 400);
  t := md(27, dbeta, 20) + (t div c4);
  t := md(7168, getvote(odely), 4000) + md(3, dphi, 8) + (t div c4);
  t := md(67, dpsi, 80) + (t div c4);
  stobroadcast(ocmdrud, t);

  lateral := 0

end;  (* LATERAL *)
GLOBAL FUNCTION PITCH:INTEGER;
(* This subroutine controls the aircraft in pitch. *)
const. armed=1; engaged=0;
begin
  p:=getvote(opitmo);
  if getvote(xreset)=1 then p:=armed;
  if (median(aglideslope)>=858) and (p=armed) then p:=engaged;
  (* Calculate deviations from nominal when glideslope is armed. *)
  if p<>engaged then
    begin
      dq:=median(aq);
      du:=median(au);
      dalpha:=median(alpha);
      dtheta:=median(theta);
      delz:=getvote(qz) + median(acmdalt);
      thrust:=0;
    end
  else (* Calculate deviations from nominal when glideslope is engaged *)
    begin
      dq:=median(aq);
      du:=median(au)+4096;
      dalpha:=median(alpha)-1678;
      dtheta:=median(theta)+634;
      delz:=getvote(qz) + md(837,getvote(qx),1000);
      thrust:=-699
    end;

  (* Calculate elevator deflection and throttle command. first elevator: *)
  t:=md(-112,dq,200) + md2(5,dalpha);
  t:=(t div C4) + md(3113,delz,100);
  t:=(t div C4) + md(220,du,500) + md(-42,dtheta,40);

  stobroadcast(ocmdele,t div C2);

  (* then throttle: *)
  t:=md1(245,dq) + md11(4739,dalpha);
  t:=(t div C8) + md6(-107,du);
  t:=(t div C2) + md12(-4058,dtheta);
  t:=(t div C4) + md2(11,delz) + thrust;

  stobroadcast(odelz,delz);
  stobroadcast(ocmdthr,t);
  stobroadcast(ocpitmo,p);

  pitch:=0
end;  (* PITCH *)
GLOBAL PROCEDURE APPINIT;
begin
  v[1]:=16384; v[2]:=16352; v[3]:=16256; v[4]:=16097;
  v[5]:=15875; v[6]:=15590; v[7]:=15245; v[8]:=14841;
  v[9]:=14378; v[10]:=13860; v[11]:=13287; v[12]:=12662;
  v[13]:=11988; v[14]:=11267; v[15]:=10502; v[16]:=9696;
  v[17]:=8852; v[18]:=7974; v[19]:=7064; v[20]:=6127;
  v[21]:=5166; v[22]:=4185; v[23]:=3188; v[24]:=2178;
  v[25]:=1159;
  c2:=2; c4:=4; c8:=8; c1024:=1024;
end. (* APPINIT,SIFTAP *)
MODULE APPLMD.SR

NAME APPLMD

TITLE SIFT: Multiple precision Multiply/Divide

These routines provide scaling functions for SIFT's applications routines

ENTRY MD, MD2, MD6, MD8, MD9, MD10, MD11, MD12, MD14

MD := (A*B)/C

MDn := (A*B)/2**n

FUNCTION MD (A, B, C: INTEGER): INTEGER;

MD
PUSHM 0,3          ; SAVE SOME REGISTERS
TRA 0,15           ; POINT AT THE DISPLAY
LOAD 1,-7,0        ; GET A
LOAD 2,-6,0        ; GET B
LOAD 0,-5,0        ; GET C
MDDO
MPY 2,1            ; PERFORM THE MULTIPLICATION
DIV 2,0            ; DIVIDE
TRA 12,3           ; STORE RESULT
POPM 0,3           ; RESTORE REGISTERS
RPS 0              ; AND RETURN

FUNCTION MD2(A, B: INTEGER): INTEGER;

MD2 := (A*B) DIV 4;

MD2
PUSHM 0,3          ; SAVE SOME REGISTERS
TRA 0,15           ; POINT AT THE DISPLAY
LOAD 1,-6,0        ; GET A
LOAD 2,-5,0        ; GET B
LOAD 0,F4          ; SET C TO 4
JU MDDO            ; GO DO IT
F4
FIX 4

FUNCTION MD6(A, B: INTEGER): INTEGER;

MD6 := (A*B) DIV 64;

MD6
PUSHM 0,3          ; SAVE SOME REGISTERS
TRA 0,15           ; POINT AT THE DISPLAY
LOAD 1,-6,0        ; GET A
LOAD 2,-5,0        ; GET B
LOAD 0,F64         ; SET C TO 64
JU MDDO            ; GO DO IT
F64
FIX 64
FUNCTION MD8(A,B:INTEGER):INTEGER;
MD8:=(A*B) DIV 256;

MD8 PUSHM 0,3 ; SAVE SOME REGISTERS
TR A 0,15 ; POINT AT THE DISPLAY
LOAD 1,-6,0 ; GET A
LOAD 2,-5,0 ; GET B
LOAD 0,F256 ; SET C TO 256
J U MDDO
F256 FIX 256

FUNCTION MD9(A,B:INTEGER):INTEGER;
MD9:=(A*B) DIV 512;

MD9 PUSHM 0,3 ; SAVE SOME REGISTERS
TR A 0,15 ; POINT AT THE DISPLAY
LOAD 1,-6,0 ; GET A
LOAD 2,-5,0 ; GET B
LOAD 0,F512 ; SET C TO 512
J U MDDO
F512 FIX 512

FUNCTION MD10(A,B:INTEGER):INTEGER;
MD10:=(A*B) DIV 1024;

MD10 PUSHM 0,3 ; SAVE SOME REGISTERS
TR A 0,15 ; POINT AT THE DISPLAY
LOAD 1,-6,0 ; GET A
LOAD 2,-5,0 ; GET B
LOAD 0,F1024 ; SET C TO 1024
J U MDDO ; GO DO IT
F1024 FIX 1024

FUNCTION MD11(A,B:INTEGER):INTEGER;
MD11:=(A*B) DIV 2048;

MD11 PUSHM 0,3 ; SAVE SOME REGISTERS
TR A 0,15 ; POINT AT THE DISPLAY
LOAD 1,-6,0 ; GET A
LOAD 2,-5,0 ; GET B
LOAD 0,F2048 ; SET C TO 2048
J U MDDO ; GO DO IT
F2048 FIX 2048
FUNCTION MD12(A,B:INTEGER):INTEGER;

MD12:=(A*B) DIV 4096;

FUNCTION MD14(A,B:INTEGER):INTEGER;

MD14:=(A*B) DIV 16384;
This report contains the software listings of the software implemented fault-tolerant computer's operating system.