A DESIGN METHODOLOGY FOR VLSI PROCESSORS

VOLUME II (Appendices A–D)

by

Joan M. Pendleton

Memorandum No. UCB/ERL M85/89

21 November 1985
A DESIGN METHODOLOGY FOR VLSI PROCESSORS

VOLUME II (Appendices A-D)

by

Joan Marie Pendleton

Memorandum No. UCB/ERL M85/89
21 November 1985

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720
A Design Methodology for VLSI Processors

Joan Marie Pendleton

ABSTRACT

A design methodology for VLSI processors has been developed. It is based on five major design levels - microarchitecture, functional block, circuit, interconnect, and process - and the interactions between them. In addition to top-down synthesis, this method formally incorporates the feedback of information from the lower design levels to the higher levels. A preliminary design phase that considers the effects of the lowest levels - circuit, interconnect, and process - on design at the highest level - microarchitecture - is described. After preliminary design, design alternates between synthesis and analysis steps as the designers proceed from the highest level to the lower levels.

SOAR (Smalltalk on a RISC), a 32 bit microprocessor designed for the efficient execution of compiled Smalltalk provides a case study of this methodology. The chip, implemented in 4 micron, single-level metal NMOS technologies, has a cycle time of 400 ns. Pipelining allows an instruction to start each cycle with the exception of loads and stores. The processor contains 35,700 transistors, is 320x432 mils, dissipates 3 watts, and is assembled in an 84-lead pin grid array package. The methodology that included a large CAD effort provided functioning chips on first silicon.
Table of Contents

Appendix A- 4 Micron NMOS Design Rules 362

Appendix B- SOAR SLANG Description 368

Appendix C- Circuit Block Logic Diagrams 471

Appendix D- SOAR Input/Output Timing Specifications 589
Appendix A
4 Micron NMOS Design Rules

Mask Layers

- Active Area or Diffusion
- N+ Implant
- Buried Contact Cut
- Polysilicon
- Contact Cut
- Metal

All units are in lambda. Lambda is 2.0 microns for this 4 micron process.
1. Single Level Rules

1.1 Active Area

Minimum width: 2.0
Minimum spacing: 3.0

1.2 Polysilicon

Minimum width: 2.0
Minimum spacing: 2.0

1.3 Metal

Minimum width: 3.0
Minimum spacing: 3.0

1.4 Contact

Minimum size: 2.0x2.0
Minimum spacing: 2.0
2. Unrelated Levels

2.1 Diffusion to Unrelated Polysilicon

Minimum spacing- 1.0

2.2 Contact to Unrelated Polysilicon

Minimum spacing- 2.0

2.3 Implant to Enhancement Transistor

Minimum spacing- 3.0
2.4 Buried Contact Cut to Unrelated Polysilicon

Minimum spacing - 1.0

2.5 Buried Contact Cut to Enhancement Transistor

Minimum spacing - 3.0

2.6 Buried Contact Cut to Depletion Transistor

Minimum spacing - 1.0

2.7 Buried Contact Cut to Unrelated Diffusion

Minimum spacing - 2.0
3. Overlaps and Enclosures

3.1 Polysilicon Overlap over Diffusion

3.2 Diffusion Enclosure of Contact Cut

3.3 Polysilicon Enclosure of Contact Cut

3.4 Metal Overlap over Contact Cut
3.5 Buried Contact Cut Enclosure of Diffusion/Polysilicon Overlap
3.6 Implant Enclosure of Diffusion/Polysilicon Overlap

Minimum buried contact enclosure
in all directions- 1.0

Minimum implant enclosure
in diffusion direction- 2.0
in all other directions- 1.0

Buried Contacts

Minimum implant enclosure
in diffusion direction- 2.0
in all other directions- 1.0

Depletion Transistors
SOAR SLANG Description

Appendix B

SOAR clock description

Pete Foley

7 nodes: masterclock, phi1, phi1+, phi2, phi2+, phi3, phi3+

phi1+ corresponds to the clock nonoverlap between phi1 and phi2. The nonoverlap clocks are for simulation purposes only.

(defnode masterclock
  (doc "masterclock, module-6 counter")
  (init (setq masterclock 5)) ; for simulation only
  (update
    (cond ((equal masterclock 5) 0)
      (t (plus masterclock 1)))))

(defnode phi1
  (doc "phase 1")
  (depends masterclock)
  (update
    (cond ((equal masterclock 0) 'ON)
      (t 'OFF)))
    ; no UNK states for clocks

(defnode phi1+
  (doc "non-overlap after phase 1")
  (depends masterclock)
  (update
    (cond ((equal masterclock 1) 'ON)
      (t 'OFF)))

(defnode phi2
  (doc "phase 2")
  (depends masterclock)
  (update
    (cond ((equal masterclock 2) 'ON)
      (t 'OFF))))
(defnode phi2+ (non-overlap after phase 2) (depends master clock) (update (cond ((equal master clock 3) 'ON) (t 'OFF))))

(defnode phi3 (phase 3) (depends master clock) (update (doc "non-overlap after phase 3") (deephase phi2+))

(defnode phi3+ (non-overlap after phase 3) (depends master clock) (update (cond ((equal master clock 5) 'ON) (t 'OFF))))

 dok

369
These OPCODE nodes provide the opcode decoding function that control PLAs on the outputs of the first and second stage control pipe latches will provide.

(defnode OPCODE1
  (depends CPIPE1s)
  (line 7 CPIPE1s<0> CPIPE1s<1> CPIPE1s<2> CPIPE1s<3>
    CPIPE1s<4> CPIPE1s<5> CPIPE1s<7>)
  (doc "gives symbolic representation of the numerically represented opcode at the output of the first stage of the control pipe")
  (update (if (numberp (plus (bits 50 CPIPE1s) (lsh (bits 77 CPIPE1s) 6)))
             (numtosymbopcode (plus (bits 50 CPIPE1s) (lsh (bits 77 CPIPE1s) 6)))
             unk)))

(defnode smOPCODE1
  (depends CPIPE1s)
  (line 2 CPIPE1s<5> CPIPE1s<7>)
  (doc "same as OPCODE1 but for jumps and calls only")
  (update (numtosymbopcode (plus (bits 5 5 CPIPE1s) (lsh (bits 7 7 CPIPE1s) 1))))
)

; the following node is mainly here to keep the pla extraction software happy

(defnode tOPCODE1
  (depends tCPIPE1s)
  (line 7 tCPIPE1s<0> tCPIPE1s<1> tCPIPE1s<2> tCPIPE1s<3>
    tCPIPE1s<4> tCPIPE1s<5> tCPIPE1s<7>)
  (doc "gives symbolic representation of the numerically represented opcode at the delayed output of the first stage of the control pipe which is decoded in tpla")
  (update (if (numberp (plus (bits 5 0 tCPIPE1s) (lsh (bits 7 7 tCPIPE1s) 6)))
             (include ;bit that differentiates jmp from call
             ))
then (numtosymbopcode (Plus (Bits 5 0 tCPIPE1s) (Lsh (Bits 7 7 tCPIPE1s) 6)))
else UNK
)
)
)

(defnode smtOPCODE1
(depends tCPIPE1s)
(line 2 tCPIPE1s<5> tCPIPE1s<7>)
(doc "same as tOPCODE but for jumps and calls only")
(update
  (If (numberp (Plus (Bits 5 5 tCPIPE1s) (Lsh (Bits 7 7 tCPIPE1s) 1)))
    then (numtosymbopcode (Plus (Bits 5 5 tCPIPE1s) (Lsh (Bits 7 7 tCPIPE1s) 1)))
    else UNK)
)
)

; Its best not to let OPCODE2 simply take OPCODE1 because of the
; possibility of jamming opcodes into the pipeline (for aborting
; instructions etc.)

(defnode OPCODE2
(depends CPIPE2s)
(line 7 CPIPE2s<0> CPIPE2s<1> CPIPE2s<2> CPIPE2s<3>
  CPIPE2s<4> CPIPE2s<5> CPIPE2s<7>)
(doc "gives symbolic representation of the numerically represented
opcode at the output of the second stage of the control pipe")
(update
  (If (numberp (Plus (Bits 5 0 CPIPE2s) (Lsh (Bits 7 7 CPIPE2s) 6))) ; include
    ; bit that differentiates jmp from call
    then (numtosymbopcode (Plus (Bits 5 0 CPIPE2s) (Lsh (Bits 7 7 CPIPE2s) 6)))
    else UNK
  )
)
)

(defnode smOPCODE2
(depends CPIPE2s)
(line 2 CPIPE2s<5> CPIPE2s<7>)
(doc "same as OPCODE2 but for jumps and calls only")
(update
  (If (numberp (Plus (Bits 5 5 CPIPE2s) (Lsh (Bits 7 7 CPIPE2s) 1)))
    then (numtosymbopcode (Plus (Bits 5 5 CPIPE2s) (Lsh (Bits 7 7 CPIPE2s) 1)))
    else UNK)
)
)

; NOTE: Bit 31 of the incoming instruction is checked by the illegal
; opcode logic to be certain it is a 0.
The easy way to do the following node is to just set this node high when OPCODE1 goes unknown, but that would not allow for expansion into a PLA.

(defun pillegalopc
  (depends OPCODE1 smOPCODE1 CPIPE1s)
  (doc "illegal opcode signal that is a direct output of illpla")
  (class illpla external)
  (update
    (or (numbtosymb (bits 9 9 CPIPE1s)) ; instr bit 31
      (not (or (memq OPCODE1
        '(flush ret0 ret1 ret2 ret3 ret4 ret5 ret6 ret7 SKIP TRAP
          load7 load6 load5 load4 load3 load2 load1 load0 store7 store6 store5 store4 store3 store2 store1 store0
          srl sra insert extract add sll sub xor and or skip
          trap1 trap2 trap3 trap4 trap5 trap6 trap7 load loadm
          loadc store storem)))
      (memq smOPCODE1 '(call jmp))))
    )
  )

(defun illegalopc
  (depends phi3)
  (update
    (if3way phi3
      pillegalopc
      illegalopc
      UNK)))

(defun Memq (item list) ; same function as LISP memq, except that
  ; it is 3 valued (UNK ON OFF)
  (cond ((unknownp item) UNK)
    ((memq item list) ON)
    (t OFF)))

;********************************************************************************
; Numeric to Symbolic opcode correspondence
;********************************************************************************
(setq numericopcenc '( (\#0104 flush) (\#0140 srl)
                         (\#0105 TRAP) (\#0151 sll)
                         (\#0106 SKIP) (\#0142 sra)
                         (\#0144 xor)
                         (\#0110 ret0) (\#0146 and)
(defun numtosymbopcode (s)
  (let ((x (assq (numeric-opcode s) numeric-opcodes)))
    (if x
        (if (numberp x) (then s else und))
        (error "opcode is given in octal"))))

; jmp really occupies opcode space o4x, o5x, o6x, o7x
; call really occupies opcode space o0x, o1x, o2x, o3x

(((defcall 0000#)
  (defjmp 0001#))
  (load 0100#)
  (load 0105#)
  (load 0104#)
  (store 0102#)
  (store 0101#)
  (load 0107#)
  (load 0106#)
  (load 0105#)
  (store 0103#)
  (store 0102#)
  (store 0101#)
  (store 0100#)
  (load 0107#)
  (load 0106#)
  (load 0105#)
  (load 0104#)
  (load 0107#)
  (load 0106#)
  (load 0105#)
  (load 0104#)
  (load 0107#)
  (load 0106#)
  (load 0105#)
  (load 0104#)
  (load 0103#)
  (load 0102#)
  (load 0101#)
  (load 0100#)
  (load 0107#)
  (load 0106#)
  (load 0105#)
  (load 0104#)
  (load 0107#)
  (load 0106#)
  (load 0105#)
  (load 0104#)
  (load 0107#)
  (load 0106#)
  (load 0105#)
  (load 0104#)
  (load 0103#)
  (load 0102#)
  (load 0101#)
  (load 0100#))

(((definsert 0156#)
  (defextract 0154#)
  (trap 0127#)
  (trap 0126#)
  (trap 0125#)
  (trap 0124#)
  (trap 0123#)
  (trap 0122#)
  (trap 0121#)
  (trap 0120#)
  (trap 0119#)
  (trap 0118#)
  (trap 0117#)
  (trap 0116#)
  (trap 0115#)
  (trap 0114#)
  (trap 0113#)
  (trap 0112#)
  (trap 0111#)
  (trap 0110#))

(0147#) (0131#) (0125#)
Instead of using the mysterious affects clause to update busses, simply define a bus as a node with infinite capacitance (memory), and list all the sources to the bus here. 

NOTE: if any of the sources to the bus should be changing while it is driving the bus (this is generally true if the source is some block of combinational logic(SXT), or if a latch is fall through), that node must be put in the depends list to insure that it is placed on the eventq and evaluated before the bus is.

---

**busD**

Note: master of PC opens on phi3, but slave doesn't open until phi1, so PCtobusD should enable stable slave data.

**DSTtobusD** active on phi3 (phi2 for forwarding)
readPCtoA  " phi2
readTBtoA  " phi2
readSWPtoA  " phi2
lastPCtobusD  " phi3

(defun node busD
  (depends phi1 DSTtobusD readPCtoA readTBtoA readSWPtoA
   lastPCtobusD ) ; muxedDST PCs lastPCs TBs omitted
  (doc "The D bus, precharged on phi1, mainly used as a result write path")
  (update
    (conflict phi1 DSTtobusD readPCtoA readTBtoA readSWPtoA ; only one can
     lastPCtobusD ) ; be busy at once
    (If3way phi1
     -1 ; precharge
    (If3way DSTtobusD ; no need for precharge check, strong drivers
     muxedDST ; write ALU result or forward
     (If3way readPCtoA
      (Logand busD PCs) ; read PC
;use Logand as bus tie is only
;a pulldown
(If3way lastPCtobusD
    lastPCs ;save PC on a call
    ;strong drivers!!!!!!!
    (If3way readTBtoA
        (Logand busD TBs) ;read TB
        (If3way readSWPtoA
            (Logand SWPs busD);read SWP
            busD ;retain old value
            UNK
        )
        UNK
    )
    UNK
)  
UNK  
UNK  
UNK  
UNK  
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
(If3way phi1
  -1 ;precharge
  (If3way busDtobusB ;strong drivers
    (Comp busD) ;reg write
  (If3way readRFaccessB
    (If (numberp RFinmuxB)
      then
        (Logand busB (Comp (rf RFinmuxB))) ;Reg read
      else UNK)
  (If3way LOADLtobusB ;write LOAD/PTRtoREG store data
    ;into reg file
    (Comp LOADLs) ;strong drivers!!!
    busB ;retain old value
    UNK)
    UNK)
  UNK))
)

******************************************************************
; bus A
; XREG or SRC1 bus through register file
;
; readRFaccessA active on phi2
; busDtobusA " phi3
; SHBtobusA " phi2
; SHAtobusA " phi2
; LOADLtobusA " phi2,phi3
; busStobusA " phi2
;
******************************************************************

(defnode busA
  (depends phi1 readRFaccessA busDtobusA SHBtobusA SHAtobusA
    busD busS RFinmuxA LOADLtobusA busStobusA Alzero)
  (doc "the A bus, precharged on phi1, mainly used for operand reads/writes ")
  (update
    (conflict phi1 busDtobusA SHBtobusA SHAtobusA LOADLtobusA busStobusA)
    (If3way (And readRFaccessA
      LOADLtobusA)
      (warning "illegal source to busA during read")
      OFF UNK)
  (If3way phi1
    -1 ;precharge
    (If3way Alzero
      0
      (If3way busDtobusA
        busD ;reg write
        UNK
        UNK)))
(If3way SHBtobusA
  (Logand busA SHB) ;read shadow register
  (If3way busStobusA
   (Logand busS busA)
   (If3way SHAtoA
    (Logand busA SHA)
  (If3way readRFaccessA
   (If (numberp RFinmuxA)
    then (Logand busA (rf RFinmuxA))
   else UNK)
  (If3way LOADLtobusA
   LOADLs
   ;strong drivers!!!
   busA
   ;retain old value
   UNK)
   UNK)
   UNK)
   UNK)
   UNK)
))))

;**********************************************************************
;
; Bus L
;
; This is the bus on which data leaves or enters the processor.
; Ideally, the A or B bus should be used for this, with data
; entering and leaving on the left side of the register file array.
; This scheme would be OK if it weren't for the nasty store instruction.
; Stores require a SRC , a register, and an immediate offset all
; to be read out during phi2 of the second cycle. The only way to
; accomplish this is to have the A bus provide the SRC to the SRC
; latch, the B bus provide the register to the INB latch, and a third
; bus (the L bus) provide the immediate date to the INB latch.
;
; SXTtobusL active on phi2,phi3
; LOADLtobusL active on phi2-phi3 ;for stores
;
;**********************************************************************

(defnode busL
  (depends SXTtobusL LOADLtobusL phi1 phi2) ;SXT LOADLs omitted
  (doc "the L bus, precharged on phi1, used for getting data in and out
       of the main data path")
  (update
(conflict phi1 SXTtobusL LOADLtobusL)
(If3way phi1
 -1 ; precharge
 (If3way SXTtobusL
   (Logand SXT busL) ; immediates (on phi2)
   (If3way (And Alzeroforce phi2) ; calls, jumps
    (Logand DIL busL)
   (If3way LOADLtobusL
    (Logand busL LOADLs) ; stores
     busL ; retain
     UNK
   )
   UNK
 )
 )
)
)
)
)
)
)
)
)
)
)
)
)
)

*******************************************************************
SBUS
S bus is used to get special registers CWP, PSW, shDST & shOPC
into the datapath on reads. It dumps onto the A bus. These special
registers can therefore be used with immediates.

(defnode busS
 (depends readCWPtoA readPSWtoA phi1) ; PSW, CWPm shOPC, shDST
 ; omitted
 (doc "special register read bus")
 (update
  (If3way phi1
   -1 ; precharge
   (If3way readCWPtoA
    (Logand busS (Lsh CWPm 4))
   (If3way readPSWtoA ; get PSW shOPC and shDST
    (Logand busS
     (Logor (Lsh PSW 5)
      (Logor (Lsh shOPC 8) shDST)))
    busS
    UNK
   )
  )
 )
 UNK
)
Databus In

Note: the phi3 dependence is not really necessary, it is just included because read data is not expected to become valid until some time in phi3.

(defnode IN
  (depends phi3 RD_WR extaddrmux phi1 phi2)
  (update
   (If3way (And phi3 RD_WR)
     (memory_read extaddrmux)
     IN
     UNK)))

(defnode DATABUSin
  (depends phi1 phi2 phi3 RD_WR busL IN)
  (update
   (If3way (Not RD_WR)
     busL
     (If3way RD_WR
       IN
       DATABUSin
       UNK)
     UNK)
   )
  )
  )
;******************************************************************
; ALU input latches (A and B)
;
; INBm now feeds the alu logic (the slave latch has been bypassed)
;
; don't open up INBm during load/store multiple so that an arbitrary
; EA decrement can be retained (very useful for Bit BLT)

(define INBm
  (depends phi3 phi2 busL busB busLtoINB busBtoINB)
  (doc "ALU input register B master. Loaded on phi2, refreshed from slave
        on phi3. If SXT is being dumped onto the L bus, then INB must be
        the destination ")
  (update
    (conflict busLtoINB busBtoINB)
    (If3way phi3
      INBm ; refresh
      (If3way busLtoINB
        busL ; take immediate data, or new pc value (-1) on calls
        ; and jumps
        (If3way busBtoINB
          (Comp busB); take read data and true the inverted bus
          INBm ; retain old value
          UNK)
        UNK)
      UNK)))

(define INAm
  (depends phi3 busA busAtoINA busDtoINA busD)
  (doc "ALU input register A master, loaded on phi2, refreshed on phi3")
  (update
    (conflict busAtoINA busDtoINA)
    (If3way phi3
      INAm ; refresh
      (If3way busAtoINA
        busA ; take read data
        (If3way busDtoINA
          busD
          INAm
          UNK))
      UNK))
The DST latch is loaded from the ALU on phi3. It must source write date during the following phi3. So master loads on phi3, slave loads on phi1, refresh on phi2.

The DST latch provides the register address for PTR to REGISTER addressing in the CWP logic.

(defnode DSTm
  (depends ALU phi3 phi2) ; WAIT DSTs omitted
  (doc "Destination latch. Holder of ALU results generated during phi3, to be written to the register file during the following phi3")
  (update
    (If3way phi2
      DSTs ; refresh
      (If3way (And phi3 (Not WAIT))
        ALU
        DSTm
        UNK)
      UNK)))

(defnode DSTs
  (depends phi1) ; DSTm omitted
  (update
    (If3way phi1
      DSTm ; take master date
      DSTs
      UNK
    )))

(defnode muxedDST
  (depends nilonreturn DSTs)
  (doc "mux which selects between the DSTs and the nill value to be driven onto the D bus through the DST push pull drivers")
  (update
    (If3way nilonreturn
      (Lsh 11 28)
      DSTs
      UNK)))))

; The A and B bus shadow registers
The B bus shadow register should also shadow the L bus when
an immediate value is used.

(defnode SHB
  (depends busB phi1 busL busLtoSHB busBtoSHB)
  (doc "bus B shadow register, master loaded on phi2 or phi3,
        refresh on phi1")
  (update
   (conflict busLtoSHB busBtoSHB)
   (If3way phi1
    SHB  ;refresh
    (If3way busLtoSHB
     busL  ;take bus data
     (If3way busBtoSHB
      (Comp busB)  ;remember B is an inverted bus!
      SHB  ;retain
      UNK
     )
    UNK
    UNK
   )
  )

(defnode SHA
  (depends busA phi1 busAtoSHA)
  (doc "bus A shadow register, master loaded on phi2 or phi3,
        refresh on phi1")
  (update
   (If3way phi1
    SHA  ;refresh
    (If3way busAtoSHA
     busA  ;take bus data
     SHA
     UNK
    )
    UNK
   )
  )

;*****************************************************

; The destination field shadow latch
; note: shDST takes DST1s during phi2 as per the generation of busSHADOW
;

(defnode shDST
  (depends phi1 busSHADOW DST1s writetoPSW) ;RESET DSTs omitted
  (doc "Destination pipe shadow register")
  (update
   (conflict busSHADOW writetoPSW)
   (If3way RESET ;slang hack only so that the "PSW" can be
     ;read and not be UNK. The node does not
     ;actually depend on RESET!!!!
     0
     (If3way phi1
       shDST ;refresh
       (If3way busSHADOW
         DST1s
         (If3way writetoPSW
           (Bits 4 0 DSTs)
           shDST
           UNK
         )
         UNK
       )
       UNK
     )
   )
  )
)

;*********************************************************

THE OPCODE SHADOW LATCH
;
; shOPC takes CPIPE1s on phi2 as per the generation of busSHADOW
; grabs bit 30, the % bit, and the 6 bit opcode (same as the lower
; 8 bits of the 9 bit CPIPE1)

(defnode shOPC
  (depends phi3 CPIPE1s busSHADOW) ;RESET omitted
  (update
   (If3way RESET
    0 ;set shOPC and shDST to something initially
    ;so that when the "PSW" is read it will not
    ;be UNK. Purely a SLANG hack. The node
    ;does not physically depend on RESET!!!!
    (If3way phi1
      shOPC ;refresh
      (If3way busSHADOW
        CPIPE1s
        (If3way writetoPSW
         (Bits 4 0 CPIPE1s)
         shOPC
         UNK
        )
        UNK
      )
    )
  )
)

;**********************************************************
(If3way phi1
  shOPC ;refresh
  (If3way busSHADOW
    (Bits 7 0 CPIPE1s) ;don't shadow immediate bit
    shOPC
    UNK
  )
  UNK
  )
)
)

;**************************************************
; The Data Input and Load data input latches
;
; The DIL holds immediate values, and in the case of call or
; jmp, the 28 bit PC value to be loaded into the PC via the
; ALU
;
; We could pull the upper 21 bits of the DIL from the CPIPE1
; SRC and DST latches (this depends on routing). But since the
; DIL LOADL and SXT are going to be physically placed on the
; left side of the register file a full 28 bit DIL is provided.
;
; since SXTtobusL is only active during phi2 and the only place
; the DIL goes to is the SXT, it is ok for this latch to be
; master only. It shouldn't matter if the
; hardware has a master-slave latch.

(defvar DIL
  (depends CPIPE1load phi2 DATABUSIn)
  (doc "Master latch of data/immediate input latch")
  (update
   (If3way phi2
    DIL ;refresh
    (If3way CPIPE1load
     DATABUSIn
     DIL ;retain
     UNK
     )
    UNK
    )
   )
  )
)
A separate latch for incoming LOAD data must be provided so that
Instruction and immediate data will not be overridden.
Since incoming data is latched on phi3 and written to the register
file on the following phi3, this latch is followed by a phi1 slave
so that data will not change during the write into the reg file.

(defnode LOADLm
  (depends DATABUSin phi1 busB DATABUSintoLOADL phi2 WAIT)
  (update
    (If3way phi1
      LOADLs1 ;refresh
      (If3way DATABUSintoLOADL
        DATABUSin
        (If3way (And phi2 (Not WAIT)) ;busBtoLOADL
          (Comp busB) ;get possible PTRtoREG data
          LOADLm ;retain
          UNK)
        UNK)
      UNK)))

(defnode LOADLs1
  (depends phi3 WAIT LOADLm)
  (update
    (If3way (And phi3 (Not WAIT))
      LOADLm
      LOADLs1
      UNK)))

(defnode LOADLs
  (depends phi1) ;LOADLm omitted
  (update
    (If3way phi1
      LOADLs1 ;take master
      LOADLs ;retain
      UNK)))

********************
;SXT sign extend bit 11 of Data Input Latch (DIL)
;
; NOTE: SXT control signals are critical path signals
;
(defnode SXT
  (depends DIL CPIPE1s storeSXT)
  (doc "sign extend 12 bit immediate held in Data Input Latch"
(DIL), or pass fast shuffle address on through, strictly combinatorial")

(update
  (If3way storeSXT
    (Logor (Lsh (Bits 22 19 DIL) 28)
      (Logor (Bits 6 0 DIL)
        (If (eq (Bits 18 18 DIL) 1)
          then (Lsh 2097151 7)
          else 0)))))
    (Logor (Lsh (Bits 11 8 DIL) 28)
      (Logor (Bits 6 0 DIL)
        (If (eq (Bits 7 7 DIL) 1)
          then (Lsh 2097151 7)
          else 0))));
; sign extend bit 7 and
; move the upper 4 bits of the constant
; into the tag bits
UNK
)}

;***************************************************************
; PC Program Counter must be master-slave because of the feedback
; path through the increneter. Master takes data on phi3
; (this is because the ALU can load the PC), slave takes data
; on phi1, refresh on phi2. Also note that the PC can drive
; the D bus on phi2 (for relative address calculations) and on
; phi3 (for saving the PC value on calls).

; NOTE: there is nothing to prevent both the D bus and the ALU
; from inputting simultaneously into the PC! e.g. the user
; could be writing to the PC as a general purpose register,
; at the same time an effective addr is being written.
ADD R17,R0,R0
JMP addr

; will crash the PC!

(defnode PCm
  (depends busD ALU ALUtoPC PCIncr phi2 writetoPC RESET phi3) ; PCs omitted
  (doc "master latch of program counter, master takes ALU or PC+ 1
       on phi3, slave takes master on phi1,
       refresh on phi2")
  (update
(conflict RESET ALUtoPC writetoPC PClnc)
(If3way RESET
  #x0ffffff)
(If3way phi2
  PCs ;refresh
  (If3way ALUtoPC
   (Bits 27 0 ALU)
   (If3way writetoPC
    (Bits 27 0 busD)
    (If3way PClnc
     (Bits 27 0 (Plus PCs 1));else PC<--PC+ 1
      PCm ;retain
      UNK
    )
    UNK
  )
  UNK
)
PCm
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
)
lastPCs ;refresh
    (If3way (And phi3 lastPCLoad)
      PCs
      lastPCm ;retain (Interlocked)
      UNK)
    UNK))
)
)

;***********************************************************************
;MAL Memory Address Latch master feeds pads, master can take either
; PCs, ALU, or TRAP vector. Slave takes master on phi1, refresh
; occurs on phi2

defnode trap
    (class cplal external)
    (depends OPCODE1)
    (update
      (Memq OPCODE1 '(TRAP))))

defnode MALm
    (depends PCtoMAL phi1 trap phi2 phi3 ALUtoMAL PCm ALU);TBs TRAPreason
    ;MALs opcmux omitted
    (doc "Memory address latch master, can take either PCs, ALU, or
         TRAP vector on phi3. Slave takes master on phi1, refresh
         occurs on phi2")
    (update
      (conflict phi2 ALUtoMAL PCtoMAL)
      (If3way phi2
        MALs ;refresh
        (If3way (And trap phi3)
          (Logor (Lsh (Bits 27 10 TBs) 10)
            (Logor (Lsh TRAPreason 6)
              (Bits 5 0 shOPC)))) ;form trap vector
          ;[TB base:reason:opcode]
          (If3way ALUtoMAL
            (Bits 27 0 ALU) ;take EA
            (If3way PCtoMAL
              ..."
PCm ; take PC + 1 (master !)
MALm ; hold old value
UNK
)
UNK
)
UNK
)
UNK
)
)

(defnode MALs
(depends MALm)
(update
MALm))

***********************************************************
;TB Trap vector base address. This is necessarily a pseudo-static
; latch because of the need to hold its value over a long period
; of time. This is in addition to making all latches pseudo
; static for chip testability (& ability to WAIT the chip).
; Master loaded on phi3, slave loaded on phi1, refresh on phi2.

(defnode TBm
(depends phi2 busD writetoTB) ; TBs omitted
(doc "Trap vector base address latch master. Loaded on phi3 from
the D bus, slave loaded on phi1 from the master, refresh
occurs on phi2")
(update
(If3way phi2
  TBs ; refresh
  (If3way writetoTB
    (Lsh (Bits 31 10 busD) 10) ; load new base addr
    TBm ; retain
    UNK
  )
  UNK
  )
)
)

(defnode TBs
(depends phi1) ; TBm omitted
(update
(If3way phi1
  TBs ; refresh
TBm ;take master
TBs ;retain
UNK
)
)

 PSW  Program Status Word  
; PSW is currently only 2 bits  
; PSW<0> Software Interrupt Enable  
; PSW<1> External Interrupt Enable  

(defnode PSW
 (depends phi2 writetoPSW phi3 trap RESET) ;DSTs enableINTS omitted
 (update
 (conflict writetoPSW enableINTS trap RESET)
 (If3way
 phi2
 PSW ;refresh
 (If3way
 writetoPSW
 (Bits 6 5 DSTs)
 (If3way enableINTS
 (Logor PSW 2)
 (If3way (And phi3 trap)
 (Logand PSW 1) ;disable interrupts, we don't care
 ;about the other 30 bits
 (If3way (And phi3 RESET) ;disable on power up
 0 ;can't use logand as above as logand
 ;with UNK is UNK
 PSW
 UNK
 )
 UNK
 )
 UNK
 )
 UNK
 )

 ;****************************************************************************
 ; SWP  Saved Window Pointer. Full 28 bit pointer
Master open on phi3, slave takes master on phi1, refresh on phi2
SWP is never read on phi2, but is saved into a reg file register
on phi3 and is written into from the DST latch on phi3

(defnode SWPm
  (depends busD phi2 writetoSWP) ;SWPs omitted
  (doc "master latch of saved window pointer.")
  (update
    (If3way phi2
      SWPs ;refresh
      (If3way writetoSWP
        busD
        SWPm ;retain
        UNK
      )
      UNK
    )
  ))

(defnode SWPs
  (depends SWPm)
  (update
    SWPm))

;*****************************************************************************
;Pointer to Register detect logic
;  This logic consists of a subtractor to compare the SWP and the MALs
;  (MSWP <27:4> - MALs<27:4> - 1)<27:7> = 0 and MALs<3> = 1
;  This signal must be valid prior to phi3!!!
;
(defnode pPTRtoREG
  (depends MALs SWPs )
  (doc " Pointer to Register detection logic. Compares MAL to SWP using
    a subtractor, checks to see if contents of MAL is a context
    object")
  (update
    (If (or (unknownp MALs)(unknownp SWPs)) then UNK
    else
      (If (and (equal (Bits 23 3
        (diff (Bits 27 4 SWPs)
        (Bits 27 4 MALs)
      1))
        (equal (Bits 3 3 MALs) 1))
        0)
      (equal (Bits 3 3 MALs) 1))
    )
  )
(defnode PTRtoREG
  (depends phi2)
  (update
   (If3way phi2
    pPTRtoREG
    PTRtoREG
    UNK))))

************************************************************
; BYTE INSERT/EXTRACT
; INSERT: Destination byte determined by bits 0 and 1 of S2
; takes upper byte of source, other bits zeroed.
;
; EXTRACT: Upper byte of destination takes byte of source determined by
; lower 2 bits of S2. Other bits zeroed.
;
; NOTE: that the master outputs of the input latches are used. This allows
; data to setup to the carrychain before phase three.
;
(defnode byteEX/INS
  (depends INAm INBm byteEX byteINS EX_INSpass )
  (doc "Upper word of byte extractor/inserter, strictly combinatorial,
       passes data through unchanged if EX/INSpass == 1")
  (update
   (conflict byteEX EX_INSpass )
   (setq byteno (Plus (Bits 00 INBm) (Times (Bits 11 INBm) 2)))
   (If3way EX_INSpass
    INAm ;pass
    (If3way byteEX
     (Bits (Minus (Times (Plus 1 byteno) 8) 1)
      (Times byteno 8) INAm)
     ; Extract to low byte, upper three bytes zeroed
     (If3way byteINS
      (Lsh (Bits 7 0 INAm) (Times byteno 8)) ; Insert low
      ; byte of INA into the specified byte of
      ; the dest register. All other bytes
      ; are zeroed
      byteEX/INS
      UNK
      )
    )
  )
)
SOAR alu description

xx nodes

This is borrowed from the RISCII description with significant changes. Inputs are taken from the byte extractor/inserter as well as the INB input latch.

Note:

BUS VALUES ARE ACTUAL VALUES, REGARDLESS OF BUS POLARITY.
CONTROL-SIGNAL VALUES ARE SYMBOLIC, ASSUMING POSITIVE POLARITY; ACTUAL VALUES ARE THE INVERSE OF THE SYMBOLIC ONES FOR SIGNALS OF NEGATIVE POLARITY.
busext has symbolic values.

(defnode pBIprocessed
      (class polarity 1)
      (depends selBIbar INBm)
      (update
       (If3way selBIbar
        (Comp INBm)
        INBm
        UNK))))

(defnode BIprocessed
      (depends pBIprocessed CPIPE1s)
      (update
       (Logand (Comp (Lsh (Bits 6 6 CPIPE1s) 31)) pBIprocessed)))

(defnode Alprocessed
      (depends byteEX/INS)
      (update byteEX/INS))
(defnode carrychain
  (doc "the carry-chain status: (CIN0 AI BI); ON: prech.; other: used")
  (depends prechCCH aluCINbar AIprocessed BIprocessed)

; the value of this node may be ON, UNK, or a list of
; length 3. It may be ON for a fully precharged
; carry chain, (ON AI BI) or (OFF AI BI) for
; a carry-chain that has been discharged by the
; corresponding inputs, or UNK for a carry-chain
; containing garbage.
(update
  (cond ((and (onp prechCCH) (onp aluCINbar))
    ON)
    (t
     (cond ((or (eq carrychain 'ON)
        (equal carrychain
          '(_,aluCINbar ,AIprocessed,BIprocessed))
        (equal carrychain
          '(ON ,AIprocessed,BIprocessed)))
        (t
          UNK))))))

(defnode aluSUM
  (doc "AIprocessed + BIprocessed + aluCIN")
  (depends AIprocessed BIprocessed CPIPE1s selaluSLL
    carrychain prechCCH aluCINbar aluCINtrue)
  ; (class polarity 1)
  (update
    (cond ((or (onp prechCCH)
      (onp (Not (Xor aluCINtrue aluCINbar))))
    (not (equal carrychain
      '(_,aluCINbar ,AIprocessed,BIprocessed))))
      'UNK)
    (t
     (If (and (onp selaluSLL) (equal (Bits 6 6 CPIPE1s) 1)) ;leave
      ;bit 31 0 on a left shift of a smallint
      then (Logand (Comp (Lsh 1 31)) (Plus (Plus
        AIprocessed BIprocessed))

(symbtonumber aluCINtrue))
  else (If (and (numberp AIprocessed)(numberp BIprocessed))
    then (+ AIprocessed BIprocessed ; use + so that a
      ; fixnum will always result
      (symbtonumber aluCINtrue))
    else UNK)))))))

;-----------------------------------------------

(defun aluZout
  (doc "signal which indicates aluresult is zero. Used as an
        input into condpla")
  (depends ALU)
  (update
    (Equal ALU 0)
    )
)

;-----------------------------------------------

;----- ALU, ALUtoD
;-----------------------------------------------

(defun pALU
  (doc "the output of the ALU (combinational node -- no latch")
  (depends aluSUM AIprocessed BIprocessed shiftAbus30 shiftAbus31
    selaluSUM selaluXOR selaluOR selaluAND aluselSR
    passALU)
  (update
    (conflict selaluSUM selaluXOR selaluOR selaluAND aluselSR
      passALU)
    (If3way
      selaluSUM
      aluSUM
      (If3way
        selaluXOR
        (Logxor AIprocessed BIprocessed)
        (If3way selaluOR
          (Logor AIprocessed BIprocessed)
          (If3way selaluAND
            (Logand AIprocessed BIprocessed)
            (If3way aluselSR ; Or of selaluSRL selaluSRA
              (Logor
                (Logand (Comp (Lsh 3 30)) ; just to be safe
                  (Lsh AIprocessed -1))
                (Logor (Lsh (symbtonumber shiftAbus30) 30)
                  (Lsh (symbtonumber shiftAbus31) 31)))))
        )
      )
    )
  )
)
(If3way passALU ; pass byte insert/ext data
  ALprocessed
  UNK
  UNK)
  UNK)
  UNK)
  UNK)
  UNK))

(defun ALU
  (depends pALU CPIPE1s)
  (update
   (Logand (Comp (Lsh (Bits 6 6 CPIPE1s) 31)) pALU)))

; ---- control of CARRY-CHAIN
; ---------------------------------

(defun prechCCH
  (doc "control: precharge the carry-chain")
  (depends phi2 phi3)
  ; (class polarity 1)
  (update
   (If3way phi2
     ON
     (If3way phi3
      OFF
      prechCCH
      UNK)
      UNK)))

; Since there is no carry PSW bit, carryin must be explicitly controlled
; by the opcode
;
(defun aluCINbar
  (doc "ALU CINbar. Must be high while prechCCH. Slang val. = real val."
  (class xcplal external)
  (depends phi3 OPCODE1 smOPCODE1)
  (update
   (Or (Not phi3) (Not (Or (Memq OPCODE1 '(sub loadm storem
      load7 load6 load5 load4 load3 load2 load1
      store7 store6 store5 store4 store3 store2 store1
      skip trap1 trap2 trap3 trap4
      trap5 trap6 trap7))))
(Memq smOPCODE1 '(call jmp)))
)
)

(defnode aluCINtrue
  (doc "ALU carry-in. Whatever when prechCCH; (Not aluCINbar) when op")
  (depends aluCINbar)
  (update
   (Not aluCINbar)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;; control of ALU OUTPUT-MUX
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defnode selaluSUM
  (doc "control: select aluSUM for output of the ALU (default mode")
  (class xcpla1 external)
  (depends EX_INSpass selaluXOR selaluOR selaluAND aluselSR)
  (update
   (Not (Or (Not EX_INSpass) selaluXOR selaluOR selaluAND aluselSR)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SOAR register file and register file decoding

Register window decoding function

Registers that formerly were accessed using the GET and PUT instructions, are now treated as general purpose registers (with some limitations).

The CWP, SWP, PC, PSW, shDST, SHA, SHB, and TB are assigned global register numbers as follows:

- "R0" 16  read/write
- PC 17  read/write
- SHB 18  read/write
- SHA 19  read/write
- SWP 20  read/write
- TB 21  read/write
- CWP 22  read/write
- shDST PSW [OPC] 23  read/write (read OPC only)

while regdecode can handle any reg # between 0 and 31, the selection of CWP-1 has already been done so that regdecode should now only decode regs > 15 and between 8 and 15.

(defun regdecode (wn rn)
  ; This function takes a window number (0,...,7) and a register number (0,...,31), and translates them into a "physical register number" (0,...,79).
  ; (8 * 8 windows) + 16 globals = 80
  ; The registers are:
  ; 15,...,8: overlap with caller (highs)
  ; (caller window is current window + 1 modulo 8);
  ; 7,...,0: overlap with callee (lows);
  ; 31,...,16: global.

  (cond ((or (not (numberp rn)) (lessp rn 0) (greaterp rn 31)) ; bad rn 'UNK)
        ((greaterp rn 15) ; global
         (diff rn 16))
        ((or (not (numberp wn)) (lessp wn 0) (greaterp wn 7)) ; bad wn 'UNK)
        ((and (eq wn 7) (greaterp rn 7))
         (plus rn 8))
        ((lessp rn 16) ; window
Note that SOAR has no locals!

;--------------------------------------------
;----- RAdecoded, RBdecoded, RDdecoded
;--------------------------------------------

; the selection of CWP or CWP-1 is done here based on bit 3 of the
; incoming reg# to more closely model the physical implementation.

(defnode RBdecoded
  (doc "the decoded reg-B address; may be: UNK, 0, 1, ..., 79")
  (depends SRC2m1 CWPm decodeEA DSTm cwpmMinus1)
  (update
    (If 3way decodeEA ;if on, then load EA incase of possible
      ;ptr to reg store
      (If (unknownp DSTm) then UNK
        else (regdecode ;don't bother checking to see if DSTm<3> = 1
          ;as the CWP flip is automatic
          (Bits 6 4 DSTm)
          (Bits 3 0 DSTm)))
    (If (or (unknownp CWPm)(unknownp SRC2m1)) then UNK
      else (regdecode
        (If (eq (Bits 3 3 SRC2m1) 1)
          then CWPm
          else cwpmMinus1) ;select CWP-1 for lows
        (If (eq (Bits 4 4 SRC2m1) 1)
          then SRC2m1 ;global
          else (+ (Logand SRC2m1 #o27) 8)))) ;add 8 if
      ;0 <= SRC2m1 < 7
      UNK
    )
  )
)

(defnode RAdecoded
  (doc "the decoded reg-A address; may be: UNK, 0, 1, ..., 79")
  (depends SRC1m1 CWPm cwpmMinus1)
  (update
    (If (or (unknownp CWPm)(unknownp SRC1m1)) then UNK
      else (regdecode
        (If (eq (Bits 3 3 SRC1m1) 1)
          then CWPm
          else cwpmMinus1)
        (If (eq (Bits 4 4 SRC1m1) 1)
          then SRC1m1
          else cwpmMinus1))
    )
  )
)
then SRC1m1  ;global
else (+ (Logand SRC1m1 #o27) 8)))  ;add 8 if
;0 <=SRC1m1 < 7
)
)

(defnode RDdecoded
(doc "the decoded DESTreg address; may be: UNK, 0, 1, ..., 79")
(depends DST2s CWPs decodeEA DSTs cwpMinus1)
(update
(If3way decodeEA  ;if on, then load EA incase of possible
 :ptr to reg store
 (If (unknownp DSTs) then UNK
   else (regdecode  ;don’t bother to make sure DSTS<3> = 1 as
     ;the CWP flip is automatic
     (Bits 6 4 DSTs)
     (Bits 3 0 DSTs)))
(If (or (unknownp CWPs)(unknownp DST2s)) then UNK
   else (regdecode
     (If (eq (Bits 3 3 DST2s) 1)
      then CWPs
      else cwpMinus1)  ;select CWP-1 for lows
     (If (eq (Bits 4 4 DST2s) 1)
      then DST2s  ;global
      else (+ (Logand DST2s #o27) 8)))  ;add 8 if
 ;0 <= SRC2s < 7
 UNK
 )))

(defnode RFinmuxA
(depends RAdecoded RDdecoded phi2 phi3)
(doc "mux to select between decoded read or write addresses")
(update
(If3way phi2
 RAdecoded
(If3way phi3
 RDdecoded
 RFinmuxA
 UNK)
 UNK)))

(defnode RFinmuxB
(depends RBdecoded RDdecoded phi2 phi3)
(doc "mux to select between decoded read or write addresses")
(update
(If3way phi2
 RBdecoded
 UNK))
(defnode regfile
  (doc "the register-file; actual contents are in the array rf")
  (depends phi3 writeRFaccess busA busB RFinmuxA RFinmuxB)
  (init (array rf t 80) ;t means type string
    (prog (i)
      (store (rf 0) 0)
      (setq i 1)
      loop ; initialize rf[i] = 1000*i + 13 ???
      (store (rf i) 'UNK)
      (setq i (1+ i))
      (cond ((lessp i 80) (go loop))))))
(update
  (If3way writeRFaccess ;only valid during phi3
    (prog
      (If (and (equal (Comp busA) busB) ;cond probably not necess
        (equal RFinmuxA RFinmuxB) ;only write to one reg!
        (numberp RFinmuxB)) ;lets be cautious
        then
        (store (rf RFinmuxB) (Comp busB))
        (store (rf 0) 0))))) ;keep "R0" 0
regfile
  (warning "REGISTER-FILE: writeRFaccess is UNK!!!")
)
)
)

; since some of the above parameters (RFinmuxA busA etc.) can have intermediate
; values during phi3 which could result in false error messages, an error node
; is created which is delayed one clock phase.
; The following nodes are for error detection in the simulation only

(defnode rfwrerr
  (depends regfile phi3)
  (update
    (If3way phi3...
(If (or (and (equal (Comp busA) busB) ; cond probably not necess
  (equal RFInmuxA RFInmuxB) ; only write to one reg!
  (numberp RFInmuxB)) ; lets be cautious
  (and (equal RFInmuxA 0)(equal RFInmuxB 0))) ; don't flag an
  ; error if the write is to RO. This way
  ; the busDtosbusA and busDtosbusB control
  ; signals won't have to include the
  ; flush opcode in their definitions.

    then OFF else ON)

OFF
UNK
)
)
)

(defvar rfwrerrcheck
  (depends phi1) ; rfwrerr omitted
  (update
    (If3way (And phi1 rfwrerr)
      (warning "Error on register file write")
      OFF
      UNK
    )
  )
)
)

(defvar SRC1equal16
  (class apla internal)
  (depends SRC1s)
  (update
    (Equal SRC1s 16))
)

(defvar readRFaccessA
  (depends phi2 ) ; SRC1specreg SRC1equal16 SRC1equalDST2 SRCvalid
  ; DSTvalid omitted
  (class apla external)
  (doc "drive the word lines with the decoded register address if
        there is no register forwarding")
  (update
    (And phi2
      (Not AIzeroforce)
      (Not (And (Not SRC1equal16)
        SRC1equalDST2
        DSTvalid SRCvalid))) ; forwarding
  )
)
(defnode SRC2equal16
  (depends SRC2s)
  (doc "when is the SRC2 pointing to R0")
  (update
   (Equal SRC2s 16)
  )
)

(defnode readRFaccessB
  (depends phi2 ) ;SRC2equalDST2, SRC2equal16, SRCvalid DSTvalid omitted
  (class apla external)
  (doc "drive the word lines with the decoded register address if
        there is no register forwarding")
  (update
   (And phi2
        (Not Alzeroforce)
        (Not (And (Not SRC2equal16) SRC2equalDST2 DSTvalid SRCvalid)))
  )
)

(defnode writeRFaccess2
  (depends OPCODE2 smOPCODE2)
  (class cpla2 external)
  (doc "drive the word lines with the decoded register address
        unless the instruction is
        one which does not do a register write in the third cycle.")
  (update
   (Not (Or (Memq OPCODE2 '(store storem ret0 ret1
                          store7 store6 store5 store4 store3 store2
                          store1 store0 load loadc loadm
                          trap1 trap2 trap3 trap4 trap5 trap6 trap7
                          ret4 ret5 flush skip SKIP))
     (Memq smOPCODE2 '(jmp))))
  )
)

(defnode writeRFaccess
  (depends phi3)
  (update
   (And phi3
        (Or writeRFaccess2 STOREwrite))
  )
)
Function to null registers on a return (null the callee's lows)

; simulation of register nilling capability in the reg file

(defun nullregs
  (depends phi3) ; nillonreturn omitted
  (update
    (if3way (and phi3 nillonreturn)
      (prog
        (cntr)
        (setq cntr 0)
        loop
        (if (equal cntr 6) then (return)) ; null 0 through 6
        (if (numberp CWPs)
          then (store (rf (regdecode CWPs cntr)) (Lsh 11 28)) ; null
          ; regs by putting an
          ; Emeritus tag on zero.
          else (return))
        (setq cntr (plus cntr 1))
        (go loop))
    OFF
    UNK
  )
)

; CWP logic

; The CWP logic consists of a master slave main CWP
; A mux directs either bits 6-4 of the DSTs,
; CWP-1, or CWP+1 into the CWP.
; The DSTs bits are directed into the CWP when the CWP is written
; to as a general purpose register. The DSTs bits are also selected
; (instead of the CWP) for input into the decoders whenever there
; is a data access cycle (to prepare for a possible PTrtoREG load
; or store.)
; The CWPs take CWPs-1 on a call or trap; and take CWPs+1
; on a ret or reti. Loading the CWPs for these instructions occurs on
; phi3 of the second cycle of the instruction. CWP must not change until
; phi1 of the third cycle as there is a result write to be done in the
; old window during phi3 of the second cycle.

(defun CWPs
  (depends writetoCWP RESET changeCWP CWPs changedCWP phi3)
(doc "master latch of CWP")
(update
(conflict changeCWP writetoCWP )
(If3way RESET ;for simulation purposes only!!!! 7
  (If3way (And phi3 (Not (Or changeCWP writetoCWP)))
    CWPs ;refresh
    (If3way changeCWP
      changedCWP
      (If3way writetoCWP ;note: this signal is already
       ;predicated on phi3
       (Bits 6 4 DSTs) ;CWP is DST field of instr
       CWPm
       UNK)
    UNK)
    UNK)
  UNK)
))

(defnode CWPs
  (depends CWPm phi2)
  (update
    (If3way (And phi2 (Not TRAP)) ;CWPstep
      CWPm
      CWPs
      UNK)))

(defnode changedCWP
  (depends CWPs changeCWP2t)
  (doc "this is the incrementer/decrementer that increments on reti,
    decrements on calls, and otherwise makes CWP-1 avaliable
    for decoding")
  (update
    (If3way changeCWP2t ; CWPincr
      (If (equal CWPs 7) then 0
       else (Plus CWPs 1))
      (If (equal CWPs 0) then 7
       else (Minus CWPs 1))
      UNK
     )
   )
)

(defnode cwpMinus1
  (depends CWPs)
  (update
    (If (equal CWPs 0) then 7
else (Minus CWPs 1)))

(defnode cwpmMinus1
  (depends CWPM)
  (update
    (If (equal CWPM 0) then 7
        else (Minus CWPM 1))))
SOAR control description

SOAR Control Description

The control pipeline in SOAR is a two stage pipeline. Each stage in the pipe consists of a master-slave pseudostatic latch which holds the opcode, the destination field, and possibly one or two other state bits. The opcode pipe latches are called CPIPE1 and CPIPE2. The output of each control pipe latch feeds a PLA which generates the control signals. The first stage PLA generates those signals that control the first two cycles of instruction execution (I-fetch and PC-RR-ALU). This stage also generates the signals that control the Load/Store memory access cycle.

The second stage PLA generates only those signals needed to control the final cycle of instruction execution.

These control PLAs are represented somewhat abstractly in the SLANG description. Control signals are individually specified, but the descriptions make use of a facility called OPCODE1 and OPCODE2. These functions decode the output of each stage of the control pipeline to indicate which instruction is in that stage of the pipe.

FLUSHING THE PIPE

For the ret instruction, the instruction currently being fetched should be flushed. (This is known during phi1 of the fetch cycle). This can be accomplished by loading CPIPE1m with the flush opcode (during phi3).

When an interrupt is taken, the currently executing instruction is allowed to finish.

(defnode CPIPE1m
  (depends DATABUSin phi3 phi1 CPIPE1load TRAPE)
  ;FLUSH1 omitted
  (doc "first stage master of two stage control pipeline. MSB bit (bit 9) is the tag bit for illegal opcode test. Bit 8 is the immediate bit, followed by bit 30, the % bit, and then the opcode. The % bit is bit 6.")
)
(update
(If3way phi2
  CPIPE1m ;refresh
  (If3way (And phi3 RESET)
    #o204
    (If3way CPIPE1load
      (Logor (Bits 30 23 DATABUSin)
        (Logor (Lsh (Bits 12 12 DATABUSin) 8)
          (Lsh (Bits 31 31 DATABUSin) 9)))
      ;take 6 bit opcode field as well as int field (bit)
      ;Bit 31 is only used for illegal
      ;opcode test
      CPIPE1m ;retain
      UNK
      )
    )
    UNK
    )
  )
)
)

;************************************************************************************
(defnode CPIPE1load
  (class cpla1 external)
  (doc "enables first stage master of control pipe to take incoming data
    Must not take incoming data during a data access cycle")
  (depends phi3 TRAP OPCODE1 WAIT RESET FLUSH1)
  (update
    (And phi3
      (Not RESET)
      (Not WAIT)
      (Not (Memq OPCODE1 '(load7 load6 load5 load4 load3 load2 load1 load0
                           store7 store6 store5 store4 store3 store2 store1 store0))))
    )
)

;************************************************************************************

(defnode CPIPE1loadc
  (depends OPCODE1 RESET)
  (class xcplal external)
  (update
    (And (Not RESET)(Memq OPCODE1 '(load loadc))))
)

(defnode CPIPE1loadm
  (depends OPCODE1 RESET)
(class xcplal external)
(update
 (And (Not RESET)
  (Memq OPCODE1 '(loadm load7 load6 load5 load4 load3 load2 load1))))

(dndef CPIPE1store
  (depends OPCODE1 RESET)
  (class xcplal external)
  (update
   (And (Not RESET) (Memq OPCODE1 '(store))))

(dndef CPIPE1storem
  (depends OPCODE1 RESET)
  (class xcplal external)
  (update
   (And (Not RESET) (Memq OPCODE1 '(storem store7 store6 store5 store4 store3 store2 store1))))

(dndef CPIPE1flush
  (depends OPCODE1 RESET)
  (class cplal external)
  (update
   (And (Not RESET)
    (Memq OPCODE1 '(ret0 ret1 ret2 ret3 ret4 ret5 ret6 ret7 TRAP))))

(dndef CPIPE1step
  (depends OPCODE1 RESET)
  (class xcplal external)
  (update
   (Or RESET
    (Not (Memq OPCODE1 '(load7 load6 load5 load4 load3 load2 load1
                         store7 store6 store5 store4 store3 store2 store1
                         ret0 ret1 ret2 ret3 ret4 ret5 ret6 ret7
                         load loadc loadm store storem TRAP))))

(dndef CPIPE1m1
  (depends phi3 phi2 phi1 CPIPE1m CPIPE1trap CPIPE1skip)
  (update
   (conflict CPIPE1step CPIPE1loadc CPIPE1store CPIPE1loadm CPIPE1storem CPIPE1flush
     (If3way phi2
      CPIPE1s
     (If3way (And CPIPE1flush phi3)
       #o206
     (If3way CPIPE1trap
       #o205
     (If3way (And CPIPE1skip (Not CPIPE1trap))
       #o206


(If3way (And CPIPE1step phi3)
  CPIPE1m
  (If3way (And CPIPE1loadc phi3)
    #o260
  (If3way (And CPIPE1store phi3)
    #o270
    (If3way (And CPIPE1loadm phi3)
      (Logor #o260 (Bits 2 0 DST1sub))
    (If3way (And CPIPE1storem phi3)
      (Logor #o270 (Bits 2 0 SRC2s))
      CPIPE1m1
    UNK)
  UNK)
  UNK)
  UNK)
  UNK)
  UNK)
  UNK)
  UNK)
  UNK))

(defnode CPIPE1s
  (depends phi1 phi3 CPIPE1m1 TRAP skipCONDvalid)
  (line 2 CPIPE1s<6> CPIPE1s<8>) ;% and immed bits
  (update
    (If3way (And phi1 (Not WAIT2))
      CPIPE1m1 ;take master
      CPIPE1s ;retaom
    UNK
    )
  )
)

; note: in the actual realization, the immed and oop/int bits won't
; be needed in the second stage of the control pipe.

(defnode CPIPE2m
  (depends phi2 phi3 WAIT CPIPEtrap)
  ;WAIT CPIPE1s omitted
  (update
    (If3way phi2
      CPIPE2s ;refresh
      (If3way CPIPEtrap
        #o204
      )
    )
    (If3way (And phi3 (Not WAIT))
      (Bits 7 0 CPIPE1s)
    )
  )
; Source and Destination Latches and Load/Store Multiple cycle
decremler

; The SRC1 and SRC2 latches each consist of two 5 bit master slave
; latches. Master is loaded on phi3, slave on phi1, refresh on phi2.
; In addition to the pads and refresh inputs to the SRC2 master, there
; is an input from the load/store multiple cycle decremler. When a
; store multiple is in progress, this input is taken on phi3 instead of
; the pads (the incoming instruction). This input is equal to SRC2s - 1.
; The store multiple instruction must then have the "seed" value
; for the store sequence in the SRC2 field.

; The DST1 and DST2 latches form a four latch set (two master slave latches)
; to delay the destination register field to the register (result) write cycle
; of the instruction. Masters are loaded on phi3, slaves on phi1, and refresh
; from associated slave to master occurs on phi2. In addition to the pads
; and refresh input to the DST1 master, there is an input from the load/
; store multiple cycle decremler. When a load multiple is in progress, this
; input is taken on phi3 instead of the pads (the incoming instruction). This
input is equal to DST1s - 1. The load multiple instruction must then have
the "seed" value for the load multiple sequence in the DST
field.

NOTE: the address 15 must be jammed into the DST pipe on calls to stuff the
PC into the appropriate register. (the CWP is decremented before the
PC is saved.)

Since the shadow registers are disabled during data access cycles, the
original operands are available should a trap occur.

Reg # 0 must always output 0 when read (hardwired to do so), & write data
to reg 0 must be ignored.

(defnode SRC1m
  (depends phi2 DATABUSin CPIPE1load) ;SRC1s omitted
  (doc "master of SRC1 master slave pair")
  (update
    (If3way phi2
      SRC1m ;refresh
      (If3way CPIPE1load
        (Bits 17 13 DATABUSin)
        ; take SRC1 field of incoming instruction
        SRC1m
        UNK
      )
      UNK
    )
  )
)

(defnode SRC1m1
  (depends CPIPE1step phi2 SRC1m phi3) ;SRC1m omitted
  (update
    (If3way (And CPIPE1step phi3)
      SRC1m
      (If3way phi2
        SRC1s
        SRC1m1
        UNK)
      UNK
    )
  )
)

(defnode SRC1s
  (depends phi1 WAIT2)
  (line 5 SRC1s<0> SRC1s<1> SRC1s<2> SRC1s<3> SRC1s<4>))
(update
    (If3way (And phi1 (Not WAIT2))
        SRC1ml
        SRC1s
        UNK))
)

(defnode DSTlmin
    (depends phi3 OPCODEl WAIT)
     (class cplal external)
     (update
        (And phi3 (Not WAIT) (Memq OPCODEl '(load7 load6 load5 load4 load3 load2 load1)))))
)

(defnode SRC2smin
    (depends phi3 OPCODEl WAIT)
     (class cplal external)
     (update
        (And phi3 (Not WAIT) (Memq OPCODEl '(storem store6 store5 store4 store3 store2 store7)))))
)

(defnode SRC2m
    (depends phi2 DATABUSin CPIPE1load) ;SRC2s omitted
    (doc "Master latch of SRC2 master slave pair")
    (update
        (If3way phi2
            SRC2m ;refresh from self
            (If3way CPIPE1load
                (Bits 11 7 DATABUSin)
                ; take SRC2 field of incoming instruction
                SRC2m
                UNK
            )
            UNK
        )
    )
)

(defnode SRC2m1
    (depends phi3 phi2 SRC2m CPIPE1step SRC2smin)
    (update
        (If3way (And CPIPE1step phi3)
            SRC2m
        (If3way SRC2smin
            SRC2sub
        (If3way phi2
            SRC2s
            SRC2m1
        )
    )
)
(defnode SRC2s
  (depends phil WAIT2)
  (line 5 SRC2s<0> SRC2s<1> SRC2s<2> SRC2s<3> SRC2s<4>)
  (update
    (If3way (And phil (Not WAIT2))
      SRC2m1
      SRC2s
      UNK)))

(defnode SRC2sub
  (depends phil SRC2s)
  (doc "this node holds the value SRC2s-1")
  (update
    (Minus SRC2s 1)
    )
)

(defnode DST1m
  (depends DATABUSin CPIPE1load phi2) ; DST1s omitted
  (doc "Master latch of DST1 master slave pair")
  (update
    (If3way phi2
      DST1m ; refresh
      (If3way CPIPE1load
        (Bits 22 18 DATABUSin)
        ; take DST field of incoming instruction
        DST1m
        UNK
      )
    UNK
    )
    )
)

(defnode DST1m1
  (depends phi3 phi2 DST1m CPIPE1step DST1min)
  (update
    (If3way (And CPIPE1step phi3)
      DST1m
      (If3way DST1min
        DST1sub
        (If3way phi2
          DST1s
        )
      )
    )
  )
(defnode DST1s
  (depends phi1 WAIT2)
  (line 5 DST1s<0> DST1s<1> DST1s<2> DST1s<3> DST1s<4>)
  (update
   (If3way (And phi1 (Not WAIT2))
    DST1m1
    DST1s
    UNK)))

(defnode DST1sub
  (depends phi1)
  (doc "this node holds the value DST1s -1")
  (update
   (Minus DST1s 1)
   )
  )

(defnode DST2m
  (depends phi1 phi3 phi2 PCstuffoncall DST2step TRAP) ;DST2s, DST1s omitted
  (doc "Master latch of DST2 master slave pair")
  (update
   (conflict PCstuffoncall DST2step trap)
   (If3way phi2
    DST2s ;refresh
    (If3way PCstuffoncall
      15 ; take PC stuff address, 15 instead of 7 as the CWP
      ; will have already changed
      (If3way (And phi3 trap) ;
        7 ; take PC stuff address for TRAP or Interrupt
        (If3way DST2step
          DST1s ; take slave of first stage
          DST2m
          UNK)
          UNK)
        UNK)))
  )

(defnode DST2s
  (depends phi1 TRAP) ; DST2m omitted
  (line 5 DST2s<0> DST2s<1> DST2s<2> DST2s<3> DST2s<4>)
  (update
   (If3way phi1
;**********************************************************
; CONTROL PIPE CONTROL SIGNALS

(defnode DST2step
  (depends phi3) ; WAIT OPCODE1 smOPCODE1 omitted
  (class cplal external)
  (doc "allows second stage of destination addr pipe to take
       first stage. Must be held up during data access cycle of
       load loadc and store")
  (update
    (And phi3 (Not WAIT)
        (Not (Memq smOPCODE1 '(call)))
        (Not (Memq OPCODE1 '(TRAP))))
  )
)

(defnode PCstuffoncall
  (depends phi3 smOPCODE1 WAIT)
  (class cplal external)
  (update
    (And phi3 (Not WAIT) (Memq smOPCODE1 '(call))(Not (Memq OPCODE1 '(TRAP))
  )
)

;**********************************************************
; SRC1 SRC2 and DST valid signals

(defnode SRCvalid
  (class apla internal)
  (depends smOPCODE1 OPCODE1 pbusDtoINA)
  (update
    (Not (Or (Memq smOPCODE1 '(jmp call)) pbusDtoINA))
  )
)

(defnode DSTvalid
  (class cpla2 external)
  (depends OPCODE2 smOPCODE2)
  (doc "indicates when the DST field of the instruction is valid")
  (update
    (Not (Or (Memq OPCODE2 '(skip trap1 trap2 trap3 trap4 trap5 trap6 trap7 trap8 trap9 trap10 trap11 trap12 trap13 trap14 trap15 trap16))
  )
)
flush store load loadc loadm storem store7 store6 store5 store4 store3
    store2 store1 store0 TRAP SKIP))
    (Memq smOPCODE2 '(call jmp))))
    ;(such as the PC) to be written to
    ; on powerup
}

;***************************************************************
; Accessing the PC as a general purpose register
;
(defnode preadPCtoA
    (class apla internal external)
    (depends SRCls SRCvalid)
    (doc "this signal is active when the SRC1 field of the instruction
        indicates the PC is to be used and the SRC1 field is valid")
    (update
        (And (Equal SRCls 17) SRCvalid)
    )
)

(defnode readPCtoA
    (depends phi2) ;preadPCtoA omitted
    (update
        (And phi2 preadPCtoA))
)

(defnode p writetoPC
    (class apla2 internal external)
    (depends DST2s DSTvalid)
    (doc "this signal is issued on phi3 when the register to be written to
        (as indicated by the DST field of the instruction) is the PC,
        and the DST2 field is valid")
    (update
        (And (Equal DST2s 17) DSTvalid)
    )
)

(defnode writetoPC
    (depends phi3) ;p writetoPC omitted
    (update
        (And phi3 p writetoPC))
)

;***************************************************************
; Register Forwarding control signals
; Forwarding is disallowed if the register to be forwarded is "RO"
; since data written to RO is to be ignored.
;
(defnode opc2load
  (class cpla2 external)
  (depends OPCODE2)
  (update
    (Memq OPCODE2 '(load0))
  )
)

(defnode SRC2equalDST2
  (depends SRC2s DST2s)
  (doc "comparator to see when SRC2s equals DST2s")
  (update
    (Equal DST2s SRC2s)
  )
)

(defnode SRC1equalDST2
  (depends SRC1s DST2s)
  (doc "comparator to see when SRC2s equal DST2s")
  (update
    (Equal SRC1s DST2s)
  )
)

(defnode pForwardtoINA
  (class apla internal)
  (doc "normal forwarding from DST latch into the A ALU input latch")
  (depends opc2load SRC1equal16 SRC1equalDST2 DSTvalid SRCvalid)
  (update
    (And (Not SRC1equal16) SRC1equalDST2
      (Not opc2load) DSTvalid SRCvalid))
)

(defnode ForwardtoINA
  (depends phi2) ; pForwardtoINA omitted
  (update
    (And phi2 pForwardtoINA))
)

(defnode pForwardtoINB
  (class apla internal external)
  (doc "normal forwarding from DST latch into the B ALU input latch.
        Data source is the LOADL")
  (depends opc2load DSTvalid SRC2equalDST2 SRCvalid SRC2equal16)
  (update
    (And (Not SRC2equal16) SRC2equalDST2
      (Not opc2load) DSTvalid SRCvalid))
)
(Not opc2load) DSTvalid SRCvalid)
)
)

(defnode ForwardtoINB
  (depends phi2) ;pForwardtoINB omitted
  (update
   (And phi2 pForwardtoINB))))

(defnode LoadforwtoINA
  (class apla1 external internal)
  (depends phi2) ;SRCequal16, SRCequalDST2, opc2load, DSTvalid,
  ;SRCvalid omitted
  (doc "this signal is active if there is register forwarding on a load
  Data source is the LOADL")
  (update
   (And phi2 (Not SRCequal16) SRCequalDST2
    opc2load DSTvalid SRCvalid)
  )
)

(defnode LoadforwtoINB
  (class apla1 external internal)
  (depends phi2) ;DST2s, SRC2equalDST2,opc2load, DSTvalid SRCvalid
  ;SRC2equal16 omitted
  (doc "this signal is active if there is register forwarding on a load")
  (update
   (And phi2 (Not SRC2equal16) SRC2equalDST2
    opc2load DSTvalid SRCvalid)
  )
)

*****************************************************************************
;
; Miscellaneous control signals
;

(defnode pbusSHADOW
  (depends OPCODE1)
  (class xcplai external internal)
  (update
   (Not (Memq OPCODE1 '(load7 load6 load5 load4 load3 load2 load1 load0
   store7 store6 store5 store4 store3 store2 store1 store0)))
 ;don't shadow during a data access cycle so the original
 ;operands will remain should a datapagefault occur
)
(defnode busSHADOW
  (depends phi2) ;pbusSHADOW omitted
  (update
   (And phi2 pbusSHADOW (Not TRAP) (numbtosymb (Bits 1 1 PSW)))))))

; RESET is Ored in here so that the MAL will take the reset PC value (0)

(defnode pALUtoPC
  (doc "Load the PC with the newPC+ 1 on calls and jumps")
  (class cplal external)
  (depends OPCODE1 smOPCODE1 RESET WAIT)
  (update
   (And (Or (Memq OPCODE1 '(ret0 ret1 ret2 ret3 ret4 ret5 ret6 ret7))
          (Memq smOPCODE1 '(call jmp))
          (Not RESET)
          (Not WAIT)))
  )
)

(defnode ALUtoPC
  (depends phi3) ;pALUtoPC omitted
  (update
   (And phi3 pALUtoPC)))

; In order to prevent direct DC paths to ground. RESET is predicated on phi2. IF a reset signal should be required on any other phase, it will have to be generated.

; NOTE: RESET must be on long enough for the PC to have transferred to the MALs, and for two instructions to have been fetched in order to fill the pipeline (DST).

(defnode sampledRESET
  (depends phi1) ;RESETin omitted
  (update
   (If3way phi1
    RESETin ;sample the input
    sampledRESET
    UNK
   )
  )
)

(defnode pRESET
  (depends phi2) ;sampledRESET omitted
(doc "RESET input to be used internally, this signal is valid from
rizing phi2 th phi2")

(update
  (If3way phi2
    sampledRESET
    pRESET
    UNK
  )
)

(defnode RESET
  (depends pRESET sampledRESET)
  (update
   (And pRESET sampledRESET))
)

(defnode pPCIncr
  (class cplal external)
  (depends RESET WAIT OPCODE1 smOPCODE1)
  (update
   (And (Not RESET)(Not WAIT)
     (Memq OPCODE1 '(flush TRAP SKIP insert extract add sub sll
                    skip trap1 trap2 trap3 trap4 trap5 trap6 trap7
                    or xor sra srl and loadO storeO))
   )
  )
)

(defnode PCIncr
  (depends phi3)
  (update
   (And phi3 pPCIncr (Not pwritetoPC))
  )
)

; NOTE: The Fast Shuffle signal should not go through the control PLA
; as it is a critical path signal.

(defnode FSHCNTL
  (doc " Fast shuffle signal, used to control the external memory addr
       mux. Should be valid phi1, phi2, and phi3.")
  (depends CPIPEls skipCONDvalid)
  (update
   (Or (numbtosymb (Bits 7 7 CPIPEls)) skipCONDvalid)
   ;Route bit 30 of the instruction
   ;out as the FSHCNTL bit
  )
)

; WAIT must be valid before phi3 rises if master latches are to be
inhibited from taking new instruction data (or the PC is to 
be inhibited from incrementing)

(defnode WAIT1
  (depends phi2)
  (update
   (If3way phi2
    WAITin
    WAIT1
    UNK)))

(defnode WAIT2
  (depends phi2)
  (update
   (If3way phi2
    WAIT
    WAIT2
    UNK)))

(defnode WAIT
  (doc "WAIT input for internal use. Valid sometime during phi2 ")
  (depends phi1 WAIT1)
  (update
   (If3way phi1
    WAIT1
    WAIT
    UNK
    )
    )
  )

;NOTE: Make sure there no restrictions to this signal!!!!

(defnode waitACK
  (doc "Wait acknowledge signal.")
  (depends phi1) ;WAIT omitted
  (update
   (If3way phi1
    WAIT2
    waitACK
    UNK
    )
    )
  )

(defnode I_D
  (depends OPCODE1)
(doc "output signal which is low when the SOAR is doing a data
access. The external fast shuffle register is conditionally
loaded on this signal.")
(class xcplal external)
(update
(Not (Memq OPCODE1 '(load7 load6 load5 load4 load3 load2 load1 load0
store7 store6 store5 store4 store3 store2 store1 store0)))
)
)

; NOTE: ANY node that is dynamic (or not fed by a signal which is
; derived from signals from pseudostatic latches) must examine
; the following node

;setq waitmax 20)
;defnode waitcnt
; (doc "counter used by dynamic nodes to determine when they should
; become UNK during a WAIT")
; (depends phi1) ;waitACK omitted
; (update
; (If3way waitACK
; (If3way phi1
; (Plus waitcnt 1)
; waitcnt
; UNK
; )
; 0
; UNK
; )
; )
;

************************************************************************
; BYTE INSERT/EXTRACT CONTROL
;
(defnode EX_INSpass
(doc "active if not byte insertion or extraction. Output of PLA1")
(class xcplal internal external)
(depend OPCODE1)
(update
(Not (Memq OPCODE1 '(extract insert)))
)
)

(defnode byteEX
(doc "high if byte extraction. Output of PLA1")

(class xcplal external)
(depends OPCODE1)
(update
  (Memq OPCODE1 '(extract)))
)
)

(defnode byteINS
  (doc "high if byte insertion. Output of PLA1")
  (depends byteEX)
  (update
   (Not byteEX)
   )
)

;**********************************************************************

(defnode pALUtoMAL
  (doc "Load the Memory Address Latch with an effective address. Also
       used to load the MAL with a CALL or JMP address + 1: the
       incrementing is done in the ALU. Active on the ALU phase (phi3.").")
  (class cplal internal external)
  (depends phi1 phi2 phi3 OPCODE1 smOPCODE1 WAIT)
  (update
   (And (Not WAIT)
   (Or (Memq OPCODE1 '(ret0 ret1 ret2 ret3 ret4 ret5 ret6 ret7))
    (Memq smOPCODE1 '(call jmp))
    (Memq OPCODE1 '(load7 load6 load5 load4 load3 load2 load1 store7 store6 store5 store4 store3 store2 store1 loadm loadc load storem store)))
   )
)

(defnode ALUtoMAL
  (depends phi3) ;pALUtoMAL omitted
  (update
   (And phi3 pALUtoMAL)))

(defnode PCtoMAL
  (class cplal external)
  (doc "default input to the MAL. IF there is no trap (or interrupt)condition,
       and the MAL is not taking an EA, then the MAL takes the PC")
  (depends phi3 ) ;pALUtoMAL WAIT omitted
  (update
   (And phi3 (Not WAIT)
   (Memq OPCODE1 '(flush SKIP load0 store0
(defnode pLOADwrite
  (class cpla2 external)
  (doc "drives both the A and B busses from the LOADL to write load data into
    the register file. Active
during register write phase (phi3).")
  (depends OPCODE2)
  (update
    (Memq OPCODE2 '(load7 load6 load5 load4 load3 load2 load1 load0)))
)

(defnode LOADwrite
  (depends phi3) ;pLOADwrite omitted
  (update
    ,(And phi3 pLOADwrite)))

(defnode pSXTtobusL
  (doc "drives the L bus with true data from the SXT to transfer immediate
    into the ALU input latch (INA)
    Active on the register read
    phase (phi2). Output of PLA1."
  (class xcplal external internal)
  (depends OPCODE1 smOPCODE1 CPIPE1s)
  (update
    ,(And (numbtosymb (Bits 8 8 CPIPE1s))
      ,(Not (Memq smOPCODE1 '(call jmp)))
      ,(Not (Memq OPCODE1 '(store7 store6 store5 store4 store3 store2 store1 store0))))
    ;don't try to drive the L bus with immediate data when
    ;your using it to output store data
    )
)

(defnode SXTtobusL
  (depends phi2) ;pSXTtobusL omitted
  (update
    (And phi2 pSXTtobusL)))

(defnode pLOADLtobusL
  (class xcplal external)
  (depends OPCODE1)
  (update
    ...)
(Memq OPCODE1 '(store7 store6 store5 store4 store3 store2 store1 store0))
)
)

;; LOADLtobusL
(defun LOADLtobusL
  (doc "drives the L bus for stores, must be active through phases phi2
       and phi3 of the store data access cycle")
  (depends phi1 pLOADLtobusL)
  (update
   (And (Not phi1) pLOADLtobusL)
  )
)

;; PTRtoREG signal must be valid before phi3!!

(defun pSTOREwrite
  (doc "drives both the A and B busses for write into the register file on
       Ptr-to-Register store. Active on the register write phase (phi3).
       Output of PLA1. Source is the LOADL ")
  (class xcpal1 external)
  (depends OPCODE1)
  (update
   (Memq OPCODE1 '(store0))
  )
)

(defun STOREwrite
  (depends PTRtoREG pSTOREwrite)
  (update
   (And PTRtoREG pSTOREwrite))
)

******************************************************************

(defun preadSWPtoA
  (class apla internal external)
  (depends SRC1s SRCvalid)
  (doc "if the SWP is in the SRC1 field of the instruction, this signal
       is enabled during phi2 ")
  (update
   (And (Equal SRC1s 20) SRCvalid)
  )
)

(defun readSWPtoA
  (depends phi2) ;preadSWPtoA omitted
(update  
(And phi2 preadSWPtoA))

(defnode pwritetoSWP  
(class apla2 internal external)  
(depends DST2s DSTvalid)  
(doc "Indicates the SWP is the target register for a write")  
(update  
(And (Equal DST2s 20) DSTvalid)  
)
)

(defnode writetoSWP  
(depends phi3) ;pwritetoSWP omitted  
(update  
(And phi3 pwritetoSWP)))

*****************************************************************
; TB (Trap Vector Base Addr) related control signals
;
(defnode preadTBtoA  
(class apla internal external)  
(depends SRC1s SRCvalid)  
(doc "TB (reg # 21) is found in the SRC1 field of the instruction")  
(update  
(And (Equal SRC1s 21) SRCvalid)  
)
)

(defnode readTBtoA  
(depends phi2) ;preadTBtoA omitted  
(update  
(And phi2 preadTBtoA)))

(defnode pwritetoTB  
(class apla2 internal external)  
(depends DST2s DSTvalid)  
(doc "TB is the target reg of a register write")  
(update  
(And (Equal DST2s 21) DSTvalid)  
)
)

(defnode writetoTB  
(depends phi3) ;pwritetoTB omitted
(defnode preadCWPtoA
  (doc "the CWP is found in the SRC1 field of an instruction")
  (class aplal internal external)
  (depends SRC1s SRCvalid)
  (update
    (And (Equal SRC1s 22) SRCvalid)
  )
)

(defnode readCWPtoA
  (depends phi2) ;preadCWPtoA omitted
  (update
    (And phi2 preadCWPtoA)
  )
)

(defnode writetoCWP
  (doc "the CWP is found in the DST field of an instruction, so it
        must take the contents of the DST reg during phi3 of the
        last cycle of the related instruction.")
  (class apla2 external)
  (depends phi3) ;DST2s, DSTvalid omitted
  (update
    (And phi3 (Equal DST2s 22) DSTvalid)
  )
)

(defnode changeCWP1
  (depends smOPCODE1 OPCODE1 WAIT)
  (class cplal external)
  (update
    (And (Not WAIT) (Memq smOPCODE1 '(call))) ;CWP decr
  )
)

(defnode changeCWP
  (depends phi3) ;changeCWP2t changeCWP1 omitted
  (doc "decrement the CWP on a call, increment on a return if the W
        option bit (opcode bit 0) is set, make no
        changes if SOAR is WAITing")
  (update
    (And phi3 (Or changeCWP1 ;CWPdecr
(defnode nillonreturn
  (depends OPCODE2)
  (class cpla2 external)
  (doc
    "nill callee's lows if the N option bit (opcode bit 1) is set
    on a return. Do the nilling during cycle 3 of the return
    instruction before the CWP slave takes the incremented value")
  (update
   (Memq OPCODE2 'ret2 ret3 ret6 ret7))
  )
)

(defnode enableINTS
  (depends phi3 OPCODE1 smOPCODE1)
  (class cplal external)
  (doc
    "reenable interrupts if the I option bit (opcode bit 2) is set
    on a return. Interrupts must be enabled as soon as possible
    upon a return as the instruction returned to may be one that
    traps")
  (update
   (And phi3 (Memq OPCODE1 'ret4 ret5 ret6 ret7)))
  )
)

; note that ptr to reg is not supported for load/store multiples

(defnode predecodeEA
  (depends OPCODE1)
  (class xcplal external)
  (update
   (Memq OPCODE1 'load loadc store))
  )
)

(defnode pdecodeEA
  (depends phi3 WAIT)
  (update
   (If3way (And phi3 (Not WAIT))
    predecodeEA
    pdecodeEA
    UNK)))
)

(defnode decodeEA
  (depends phi1 pdecodeEA)
(doc "this signal controls the muxes into the register file decoders
to select between data from the instruction or data from and
Effective Address (for possible PTR-to-REG")
(update
 (If3way phi1
   pdecodeEA
   decodeEA
   UNK
   )
 )
)

;******************************************************************************
;
; PSW related control signals
;   PSW<0>  SI enable
;   PSW<1>  Interrupt enable

(defnode preadPSWtoA
  (class apla1 internal external)
  (doc "read the PSW,shDST, and shOPC onto busA via busS")
  (depends SRC1s SRCvalid)
  (update
   (And (Equal SRC1s 23) SRCvalid)
   )
  )

(defnode readPSWtoA
  (depends phi2) ;preadPSWtoA omitted
  (update
   (And phi2 preadPSWtoA))
  )

(defnode writetoPSW
  (class apla2 external)
  (doc "write into the PSW and shDST as a general purpose register")
  (depends phi3)
  (update
   (And phi3 (Equal DST2s 23) DSTvalid)
   )
  )

;******************************************************************************
;
; DSTtobusD is also asserted during phi2 if there is register forwarding.
;
(defnode DSTtobusDa2
(depends pForwardtoINA pForwardtoINB pbusDtoINA)
(doc "output of apla that is used in forming DSTtobusD")
(class apla external)
(update
(Or pForwardtoINA pForwardtoINB pbusDtoINA); increment the Load/Store addr)
)

(defnode DSTtobusD2
(doc "enables DST latch onto the D bus. Used for normal write of ALU result into the register file.")
(depends OPCODE2)
(class cpla2 external)
(update
(Memq OPCODE2
'(add sub ret2 ret3 ret6 ret7 srl sll sra and or xor extract insert)) )
)

(defnode DSTtobusD
(depends phi3 phi2) ; DSTtobusD2, DSTtobusDa2
(doc "enables DST latch onto the D bus. Used for normal write of ALU result into the register file on phi3, and for register forwarding on phi2")
(update
(Or (And phi2 DSTtobusDa2)
(And phi3 DSTtobusD2))
)
)

(defnode RD_WR
(doc "read write control. Always high unless a store. Output of PLA1.
Need not be disabled an PTR-to-Register store as write will be to an unused portion of register file space in memory. It is also assumed that a sizeable portion of the register file space will be locked into core so that unnecessary page faults do not occur during PTRtoREG stores")
(depends OPCODE1)
(class xcpla1 external)
(update
(Not (Memq OPCODE1 '(store7 store6 store5 store4 store3 store2 store1 store0)))
)
)

(defnode busDtobusA2
(doc "enables strong bus tie drivers from bus D to bus A. Used during normal write of ALU result into the register file, and for saving the PC on calls. Active during"
register write phase (phi3). Output of PLA2.
Also active during phi2 for register forwarding and if one of
the source registers is the PC.

(deps OPCODE2 smOPCODE2)
(class cpla2 external)
(update
  (Or (Memq smOPCODE2 '(call))
      (Memq OPCODE2 'TRAP
        add sub and or xor sll srl sra
        insert extract
        ret2 ret3 ret6 ret7))
  ;ret for nilling registers)
)

(defnode busDtobusAa
  (doc "precursor to busDtobusA that is an output of apla")
  (deps preadSWPtoA preadTBtoA pForwardtoINA preadPCtoA)
  (class apla external)
  (update
    (Or preadSWPtoA preadTBtoA pForwardtoINA preadPCtoA)))

(defnode busDtobusA
  (deps phi2 phi3 TRAP) ;busDtobusAa, busDtobusA2 omitted
  (update
    (Or (And phi3 busDtobusA2)
        (And phi2 busDtobusAa))))

(defnode busDtobusB2
  (doc "enables strong bus tie drivers from bus D to bus B. Used during
      normal write of ALU result into the register file,
      and for saving the PC on calls. For these occasions
      it is active during the register write phase (phi3) and is an
      output of PLA2.
      Is also used for reading the PC for relative address calculations
      on loads and stores. Here is is active during the register read
      phase (phi2) and is an output of PLA1.
      Also used on phase2 for register forwarding.")
  (deps OPCODE2 smOPCODE2)
  (class cpla2 external)
  (update
    (Or (Memq smOPCODE2 '(call))
        (Memq OPCODE2 'TRAP add sub and or xor sll srl sra insert extract
                   ret2 ret3 ret6 ret7)) ;ret for nilling registers
))

(defnode busDtobusB
(depends phi2 phi3 TRAP) ;busDtobusB2 pForwardtoINB
(update
 (Or (And phi3 busDtobusB2)
      (And phi2 pForwardtoINB)))))

(defun lastPCtobusD
 (doc "enables slave of second latch of program counter chain onto
      bus D for PC saves on calls and traps")
 (class cpla2 external)
 (depends phi3) ;OPCODE2, smOPCODE2 omitted
 (update
  (And phi3 (Or (Memq OPCODE2 '(TRAP))
                (Memq smOPCODE2 '(call)))))
)

;******************************************************************
; Shadow register control signals
;
(defun pSHBtobusA
 (class apla1 external internal)
 (depends SRC1s SRCvalid)
 (doc "enable shadow register B onto A bus if the shadow reg is in
      the SRC1 field of the instruction")
 (update
  (And (Equal SRC1s 18) SRCvalid))
)

(defun SHBtobusA
 (depends phi2) ;pSHBtobusA omitted
 (update
  (And phi2 pSHBtobusA))
)

(defun writetoSHB
 (class apla2 external)
 (depends phi3) ;DST2s, DSTvalid omitted
 (doc "write to the SHB as a gen purpose register")
 (update
  (And phi3 (Equal DST2s 18) DSTvalid))
)

(defun busBtoSHB
 (depends writetoSHB phi2 busSHADOW pbusBtoINB)
(update
 (conflict writetoSHB SXTtobusL)
 (Or writetoSHB (And busSHADOW pbusBtoINB))
 )

(defun busLtoSHB
 (depends busSHADOW pbusLtoINB phi2)
 (update
  (And busSHADOW pbusLtoINB)
 )
)

;**************************

(defun pSHAtobusA
 (class apla1 external internal)
 (depends SRC1s SRCvalid)
 (doc "enable shadow register A onto A bus if the shadow reg is in
 the SRC1 field of the instruction")
 (update
  (And (Equal SRC1s 19) SRCvalid)
 )
)

(defun SHAtobusA
 (depends phi2) ;pSHAtobusA omitted
 (update
  (And phi2 pSHAtobusA))
)

(defun writetoSHA
 (class apla2 external)
 (depends phi3) ;DST2s, DSTvalid omitted
 (doc "write to the SHA as a gen purpose register")
 (update
  (And phi3 (Equal DST2s 19) DSTvalid)
 )
)

(defun busAtoSHA
 (depends busSHADOW writetoSHA)
 (update
  (conflict busSHADOW writetoSHA)
  (Or busSHADOW writetoSHA)
 )
)
; ALU input latch input control signals
;
; Note that this signal is very similar to CPIE1load

(defun busAtoINA
  (depends phi2) ; OPCODE1 omitted
  (update
    (And phi2 (Not pbusDtoINA))))

(defun pbusDtoINA
  (class cpla2 external)
  (doc "precursor to busDtoINA that is an output of apla")
  (depends OPCODE2)
  (update
    (Memq OPCODE2 '(load7 load6 load5 load4 load3 load2 loadm
      load1 store1 load loadc store
      store7 store6 store5 store4 store3 store2 storem))))

(defun busDtoINA
  (depends phi2) ; pbusDtoINA omitted
  (doc "take the D bus for EA calculations during a loadm or storem")
  (update
    (And phi2 pbusDtoINA)))

(defun pbusLtoINB
  (class xcpla1 external)
  (depends pSXTtobusL Alzeroforce)
  (doc "enable L bus into the B alu input latch on the read phase")
  (update
    (Or pSXTtobusL Alzeroforce)
    ; don't allow
    ; INB to take date during the data access
    ; cycles of load loadm store storem loadc )
  )

(defun busLtoINB
  (depends phi2) ; pbusLtoINB omitted
  (update
    (And phi2 pbusLtoINB)))

(defun pbusBtoINB
  (depends pbusLtoINB pbusDtoINA)
  (doc "enable B bus into the B alu input latch on the read phase")
  (update
    (And (Not pbusLtoINB)(Not pbusDtoINA)) ; don't allow
(defnode busBtoINB
  (depends phi2)
  (update
   (And phi2 pbusBtoINB)))

; The LOADL normally takes the B bus on phi2 for a possible PTR to REG load
; or store. This is the normal mode of operation because it is not
; possible to know if it is a PTRtoREG soon enough to take the B bus
; as the exceptional case. What must be done then, is to prevent the
; LOADL from taking the DATABUSin on phi3 if there is any kind of load
; or store data in the latch.

(defnode pDATABUSintoLOADL
  (depends WAIT OPCODE1)
  (class cplal external)
  (doc "signal to load DATABUSin into LOADL on phi3, overwriting whatever
        was taken off the B bus on phi2")
  (update
   (And (Not WAIT)
        (Not (Memq OPCODE1 '(store storem store7 store6 store5 store4
                               store3 store2 store1))) ; latch holds either normal
        ; store or PTRtoREG store data
   )
  )

(defnode DATABUSintoLOADL
  (depends phi3)
  (update
   (And phi3 (Not PTRtoREG) pDATABUSintoLOADL)))

(defnode storeSXT
  (depends OPCODE1)
  (class tplal external)
  (doc "tells the SXT that the immediate data is in the store instruction
        format")
  (update
   (Memq OPCODE1 '(store storem))
  )
)

;******************************************************************
; busS to busA tie control

(defnode pbusStobusA
(class aplal external internal)
(depending preadPSWtoA preadCWPtoA)
(update
 (Or preadPSWtoA preadCWPtoA))
)
)

(defun busStobusA
 (depending phi2) ;pbusStobusA omitted
(update
 (And phi2 pbusStobusA)))

;*****************************************************************
; LOADL to busA and busB tie control
(defun LOADLtobusA
 (depending LoadforwtoINA phi3) ;pLOADwrite STOREwrite omitted
(update
 (Or LoadforwtoINA (And phi3 (Or pLOADwrite STOREwrite)))))
)
)

(defun LOADLtobusB
 (depending LoadforwtoINB phi3) ;pLOADwrite STOREwrite omitted
(update
 (Or LoadforwtoINB (And phi3 (Or pLOADwrite STOREwrite)))))
)
)

;*****************************************************************
(defun selBIbar
 (class xcplal external)
(depending OPCODEl)
(doc "select the complimented output of the ALU input latch B for a subtract instruction")
(update
 (Memq OPCODEl '(sub storem loadm skip trap1 trap2 trap3 trap4 store7 store6 store5 store4 store3 store2 store1 load7 load6 load5 load4 load3 load2 load1 trap5 trap6 trap7)))
)
)

(defun selaluXOR
 (class xcplal internal external)
(depends OPCODE1)
(update
  (Memq OPCODE1 '(xor))))

(defnode selaluOR
  (class xcplal internal external)
  (depends OPCODE1)
  (update
   (Memq OPCODE1 '(or)))))

(defnode selaluAND
  (class xcplal internal external)
  (depends OPCODE1)
  (update
   (Memq OPCODE1 '(and)))))

;*******************************************************************************
;
; More ALU control signals. Unique to SOAR
;
(defnode Alzeroforce
  (doc "force 0 as the A input to the ALU to increment the new PC")
  (class apla internal external)
  (depends smOPCODE1 OPCODE1)
  (update
   (Memq smOPCODE1 '(jmp call))))

(defnode pread0toA
  (class apla internal)
  (depends SRC1s SRCvalid)
  (doc "causes register 16 to be read")
  (update
   (And (Equal SRC1s 16) SRCvalid)))

(defnode passALU
  (doc "pass inserter/extractor results through the ALU")
  (depends EX_INSpass)
  (update
   (Not EX_INSpass)
   )
)

(defnode aluselSR
  (class xcplal internal external)
  (depends OPCODE1)
  (update
   (Memq OPCODE1 '(sra srl))))
(defnode selaluSLL
  (depends OPCODE1)
  (update
   (Memq OPCODE1 '(sll))))

(defnode Alzero
  (class apla external)
  (depends Alzeroforce pread0toA phi2)
  (doc "zeroes busA")
  (update
   (And phi2 (Or Alzeroforce pread0toA))))
NOTE: the ALU performs S1 - S2 when conditions are generated

--- aluCout, aluVout

; If in tagged mode, then bits 30 of both inputs and bit
; 30 of the output are examined to see if there is a carryout.
; Otherwise, the ALU operates as a normal 32 bit ALU

CASE:
  1  int(30)   int(30)   result(30)   non %
  2  int(31)   int(31)   result(31)   %

For effective address calculations, no meaningful carryout or
overflow is generated.

The following nodes capture the information input to the condition code
pla so that these inputs will not go invalid before phi2 of the following
cycle. For clarity's sake, although a few bits of a bus are to be
captured, the slang captures the entire bus and then extracts the
relevant bits using Bits later on.

(defnode AIprocessedMSB
  (depends CPIPE1s AIprocessed)
  (update
    (If3way (numbtosymb (Bits 6 6 CPIPE1s))
      (numbtosymb (Bits 30 30 AIprocessed)))
    (numbtosymb (Bits 31 31 AIprocessed))
    UNK)))

(defnode cAIprocessedMSB
  (depends phi3 AIprocessedMSB)
  (update
    (If3way phi3
      AIprocessedMSB
      cAIprocessedMSB)
(defnode BIprocessedMSB
  (depends CPIPE1s BIprocessed)
  (update
   (If3way (numbtosymb (Bits 6 6 CPIPE1s))
    (numbtosymb (Bits 30 30 BIprocessed))
    (numbtosymb (Bits 31 31 BIprocessed))
    UNK)))

(defnode cBIprocessedMSB
  (depends phi3 BIprocessedMSB)
  (update
   (If3way phi3
    BIprocessedMSB
    cBIprocessedMSB
    UNK)))

(defnode ALUmsb
  (depends CPIPE1s ALU)
  (update
   (If3way (numbtosymb (Bits 6 6 CPIPE1s))
    (numbtosymb (Bits 30 30 ALU))
    (numbtosymb (Bits 31 31 ALU))
    UNK)))

(defnode cALUmsb
  (depends phi3 ALUmsb)
  (update
   (If3way phi3
    ALUmsb
    cALUmsb
    UNK)))

(defnode cselaluSUM
  (depends phi3 selaluSUM)
  (update
   (If3way phi3
    selaluSUM
    cselaluSUM
    cselaluSUM
    UNK
    )
   )
  )

(defnode caluZout
  (depends phi3 aluZout)
(update
   (If3way phi3
      aluZout
      caluZout
      UNK
   )
)
)

(defnode cDST1s
   (depends phi3 DST1s)
   (line 5 cDST1s<0> cDST1s<1> cDST1s<2> cDST1s<3> cDST1s<4>)
   (update
      (If3way phi3
         DST1s
         cDST1s
         UNK
      )
   )
)

;***********************************************************************

(defnode aluCout
   (depends cselaluSUM cAIprocessedMSB cBIprocessedMSB cALUmsb)
   (class condpla internal)
   (update
      (And cselaluSUM
         (Or (And cAIprocessedMSB cBIprocessedMSB)
         (And cAIprocessedMSB (Not cALUmsb))
         (And cBIprocessedMSB (Not cALUmsb)))))

(defnode aluVout
   (doc "ALU adder overflow, ANDed with selaluSUM. Also true if overflow on sll in tagged mode.")
   (class condpla internal external)
   (depends cselaluSUM cAIprocessedMSB cBIprocessedMSB cALUmsb)
   (update
      (And cselaluSUM
         (Or (And (Not cAIprocessedMSB)(Not cBIprocessedMSB) cALUmsb)
         (And cAIprocessedMSB cBIprocessedMSB (Not cALUmsb)))))

(defnode aluSout
   (class condpla internal)
(depends cALUmsb)
(update
cALUmsb))

(defnode Ain0
(depends Alprocessed)
(update
(Equal Alprocessed 0)))

(defnode cAin0
(depends Ain0 phi3)
(update
(If3way phi3
Ain0
cAin0 UNK)))

(defnode pCONDvalid
(class condpla external)
(depends aluSout caluZout aluVout aluCout cDST1s)
(doc "combinational output of condpla which indicates a valid condition")
(update
(Or (And caluZout (Equal cDST1s #o04)) ;EQ
(And (Not caluZout) (Equal cDST1s #o05)) ;NE
(And (Or (And aluSout (Not aluVout))
(And (Not aluSout) aluVout))
(Equal cDST1s #o02)) ;LT
(And (Not (Or (And aluSout (Not aluVout))
(And (Not aluSout) aluVout))
(Equal cDST1s #o03)) ;GE
(And (Or (And aluSout (Not aluVout))
(And (Not aluSout) aluVout)
caluZout)
(Equal cDST1s #o06)) ;LE
(And (Not (Or (And aluSout (Not aluVout))
(And (Not aluSout) aluVout)
caluZout))
(Equal cDST1s #o07)) ;GT
(And (Not aluCout) (Equal cDST1s #o12)) ;LTU or IN0
(And aluCout (Equal cDST1s #o13)) ;GEU or OUT0
(And (Or (Not aluCout) caluZout) (Equal cDST1s #o16)) ;LEU
(And (Not (Or (Not aluCout) caluZout)) (Equal cDST1s #o17)) ;GTU
(Equal cDST1s #o01) ;always
(And (Or (Not aluCout) caluZout) (Not cAin0) (Equal cDST1s #o22));IN1
(And (Or (Not (Or (Not aluCout) caluZout)) cAin0) (Equal cDST1s #o23));OUT1
(defnode skipCONDvalid
  (depends phi3 phi1 RESET WAIT2 pCONDvalid skipCONDenable)
  (doc "condition valid signal used internally.")
  (update
   (If3way RESET
    OFF
     (If3way phi2
      skipCONDvalid ;refresh
      (If3way (And phi1 (Not WAIT2))
        (And pCONDvalid skipCONDenable)
        skipCONDvalid
        UNK)
      UNK)
     UNK)
   UNK)
  )
)

(defnode lateskip
  (depends phi2)
  (update
   (If3way phi2
    skipCONDvalid lateskip
    UNK)))

(defnode CPIPEskip
  (depends lateskip skipCONDvalid)
  (update
   (Not (Or lateskip (Not skipCONDvalid)))))
The two external interrupts are sampled during phi2, and the sampled interrupt input is gated into the control PLAs during phi3. The interrupt sequence will start on the next phi1 unless interrupts are disabled. Interrupts are disabled by the hardware, and are not reenabled until a reti instruction is executed. The interrupt vector (which vectors SOAR into a jump table) is formed by the concatenation of the TB with the cause of the interrupt (this includes the opcode), and is loaded into the MAL.

Interrupt or Trap vector <TB>:<4 bit "cause">:<6 bit opcode>

Current "cause" encoding is:

0000 Illegal opcode
0001 Tag TRAP
0010 SWI
0011 window overflow
0100 window underflow
0101 data page fault
0110 trap instruction
0111 GS trap
1000 instruction page fault
1001 I/O request

The types of tag traps are:

;******************************************************************

SHIFT Logic

This is some of the ALU shift function that has been pushed into the trap plal to simplify the ALU. Bits 30 and 31 that result from a sra or srl in either tagged or non-tagged mode are output by the trap plal

(defnode shiftAbus30
(depends Alprocessed OPCODE1 CPIE1s)
(class tpla1 external)
(doc "bit 30 of result after a srl or sra")
(update
(Or (And (numbtosymb (Bits 6 6 CPIPE1s))) ;tagged mode
   (Memq OPCODE1 '(sra))
   (numbtosymb (Bits 30 30 Alprocessed))) ;repeat sign bit
(And (Not (numbtosymb (Bits 6 6 CPIPE1s)))) ;untagged mode
   (Memq OPCODE1 '(sra srl))
   (numbtosymb (Bits 31 31 Alprocessed)))) ;take sign bit
;note that for tagged mode srl, bit 30 is 0
)
)

(defnode shiftAbus31
  (depends Alprocessed OPCODE1 CPIPE1s)
  (class tpla1 external)
  (doc "bit 31 of result after a sra or srl")
  (update
   (Or (And (numbtosymb (Bits 6 6 CPIPE1s))) ;tagged mode
      (Memq OPCODE1 '(sra srl))
      (numbtosymb (Bits 31 31 Alprocessed))) ;retain bit 31 unchanged
   (And (Not (numbtosymb (Bits 6 6 CPIPE1s)))) ;untagged mode
      (Memq OPCODE1 '(sra))
   (numbtosymb (Bits 31 31 Alprocessed)))) ;repeat sign bit
;note that for untagged mode srl, bit 31 is 0
)
)

;*******************************************************************

; Interrupt Request handling logic
;
; The IOINT & pageINT asynch inputs are handled in the following manner.
; The input is gated on phi2 into a static flip-flop. The flip-flop
; output is sampled on phi3 (it is held on some node until the following
; phi2). The flip-flop is reset on phi1.
;
; (defnode sampledIOINT
  (doc "sampled external IO interrupt. Static flip flop that sees input
       during phi2 and is reset during phi1")
  (depends IOINTin phi1 phi2) ;PSW omitted
  (update
   (If3way phi2
      (And IOINTin (numbtosymb (Bits 1 1 PSW))) ;make sure interrupt is
       ;enabled
      (If3way phi1
         OFF
         sampledIOINT
      )
   )
  )
)
This asynchronous input is sampled during phi2
to give the cache logic time to process a page fault

; pageflNTin must be valid sometime during phi2

;********************************************************

; INSTRUCTION or DATA page fault distinguishing logic

;******************************************************************************************


(update
  (If3way phi3
   (And sampledpagefINT pbusDtoINA)
    ; to isolate data page fault
datapagefINT
    UNK
   )
  )
)
)

(defnode instrpagefINT
  (depends phi3) ; sampledpagefINT omitted
  (update
   (If3way phi3
    (And sampledpagefINT (Not pbusDtoINA))
    instrpagefINT
    UNK
   )
  )
)
)

;**********************************************************************
; The following signals are used as inputs to the trap pla, so must
; be captured by the phi2 clock for use during the next cycle
; for clarity, the entire busA and busB are captured (this makes
; the tag trap generation code easier to read), when in reality only
; the upper four bits of each bus is needed.

(defnode tCPIPEls
  (depends phi2) ; CPIPEls omitted
  (line 2 tCPIPEls<6> tCPIPEls<8>)
  (update
   (If3way phi2
    CPIPEls
    tCPIPEls
    UNK
   )
  )
)
)

; busA and busB must be captured before phi3 for use in tag
; trap detection

(defnode tbusA
  (depends busA phi2)
  (update
   (If3way phi2
    )
  )
)
\begin{verbatim}
(defun tconv (temp)
  (cond ((equal temp nil) OFF)
        ((equal temp t) ON)
        (t UNK)))

(defun itconv (temp)
  (cond ((equal temp OFF) nil)
        ((equal temp ON) t)
        (t UNK)))
\end{verbatim}
((equal temp ON) t)
(t nil))

;********************************************************************************************

(defun notanINT
  (depends tbusA tbusB tOPCODE1 tCPIPE1s)
  (class tpla internal)
  (doc "checks to see if A and B bus are both ints for the add, sub, logical,
shift, skip, and trap instructions. If Immediate mode is used,
then only the A bus is checked. If either operand is not an
int, then this node goes high")
  (update
   (or (and (not (numbtosymb (bits 8 8 tCPIPE1s)))
      (memq tOPCODE1
       '(sll srl sra add sub xor and or skip
         trap1 trap2 trap3 trap4 trap5 trap6 trap7))
      (or (equal (bits 31 31 tbusA) 1)
        (equal (bits 31 31 tbusB) 0)) ; b is inverted
      (and (numbtosymb (bits 8 8 tCPIPE1s))
        (memq tOPCODE1
         '(sll srl sra add sub xor and or skip
           trap1 trap2 trap3 trap4 trap5 trap6 trap7))
        (equal (bits 31 31 tbusA) 1)))))

(defun loadTRAP
  (depends tbusA tbusB tOPCODE1 tCPIPE1s)
  (class tpla internal)
  (doc "if the instruction is a load or loadc, assert if

  (tbusA is an integer and either
   (the instruction is an immediate or tbusB is an integer)),
or (tbusA is an OOP and the instruction is not an immediate
and tbusB is an OOP")

  (update
   (and (memq tOPCODE1 '(load loadc))
     (or (and (equal (bits 31 31 tbusA) 0) ; A is a small int (SI)
        (or (and (equal (bits 31 31 tbusB) 1) ; B is SI
           (not (numbtosymb (bits 8 8 tCPIPE1s))))
            (numbtosymb (bits 8 8 tCPIPE1s)) ; immediate
        ))
     (and (not (numbtosymb (bits 8 8 tCPIPE1s))) ; not immediate
      (equal (bits 31 31 tbusB) 0) ; B is OOP
      (equal (bits 31 31 tbusA) 1) ; A is OOP
     )))

(defun RXint
  (depends tbusA tOPCODE1)
(class tpla internal)
(doc "Since stores can only be immediate, 
check the A bus to make sure it is an oop for 
tagged mode stores. A trap occurs if the A bus is an int")
(update
(And (Equal (Bits 31 31 tbusA) 0) 
(Memq tOPCODE1 '(store)))
)
)

; RDcontext will always be on during the precharge phase since the context 
; flag is 1111 (and its tagged mode)

(defnode RDcontext
(depends tbusB tOPCODE1 )
(class tpla internal)
(doc " Check the B bus to see if 
the store data is a context. (tagged mode of course")
(update
(And (Memq tOPCODE1 '(store ))
 (Equal (Bits 31 28 tbusB) 0)) ;busB is inverted
)
)

(defnode ptagcompare
(depends busA busB)
(class tagcompla external)
(doc "comparison of busA and busB tag bits to be used in the Solder 
trap check the node is high if busB is older than busA. possible 
implemented with a subtractor.")
(update
(Or (numbtosymb (Bits 31 31 busB))
 (Not (numbtosymb (Bits 31 31 busA))))
(numbtosymb (Bits 30 30 busA))
(And (numbtosymb (Bits 30 30 busB))
 (Not (numbtosymb (Bits 30 30 busA)))
(Or (And (Not (numbtosymb (Bits 29 29 busB))))
 (Not (numbtosymb (Bits 29 29 busA))))
(And (Not (numbtosymb (Bits 28 28 busB)))
 (Not (numbtosymb (Bits 28 28 busA))))
(And (Not (numbtosymb (Bits 29 29 busB))))
(And (Not (numbtosymb (Bits 29 29 busA))))
(Not (numbtosymb (Bits 28 28 busA))))
(And (Not (numbtosymb (Bits 28 28 busB))))
(Not (numbtosymb (Bits 29 29 busB))))))))}}}
(defnode tagcompare
  (depends phi2 ptagcompare)
  (update
   (if3way phi2
    ptagcompare
    tagcompare
    UNK))
)

(defnode S1older
  (depends tOPCODE1 tagcompare)
  (class tpla internal)
  (doc "store data should be younger than S1, so check to see if busA is older than busB contents")
  (update
   (and (not tagcompare) (mem tOPCODE1 '(store))
    )
  )
)

(defnode nonLIFO
  (depends tbusA tOPCODE1)
  (class tpla internal)
  (doc "check to see if the return address is an OOP, the return addr is specified in the RM field which is read out on busA")
  (update
   (and (equal (bits 31 31 tbusA) 1)
    (mem tOPCODE1 '(ret0 ret1 ret2 ret3 ret4 ret5 ret6 ret7))
    )
  )
)

;***************************************************
; enable predicates

(defnode pov_unflow
  (class tpla external)
  (doc "opcode predicate to ov_unflow that is generated by tpla")
  (depends tOPCODE1 tCPIPE1s)
  (update
   (and (numb2symb (bits 6 6 tCPIPE1s))(mem tOPCODE1 '(sub add sl)))
    ;predicate is low if not a tagged instruction
    )
)

(defnode skipCONDenable
  (depends tOPCODE1)
  (class tpla external)
(doc "signal that indicates the current instruction should look at condition codes")
(update
  (Memq tOPCODE1 '(skip)))
)

;*******************************************************************************
;
;  WINDOW OVERFLOW/UNDERFLOW DETECT LOGIC
;
;
(defnode winoverflow
  (depends changedCWP phi3) ;changeCWP SWPs omitted
  (doc "Overflow occurs if the CWP is one greater than the SWPs and a call is about to be made")
  (update
    (If3way phi3
      (If3way changeCWP1
        (Equal changedCWP (Bits 6 4 SWPs))
        OFF
        UNK)
      winoverflow
      UNK)
    )
  )
)

(defnode changeCWP2t
  (depends OPCODE1 smOPCODE1 WAIT)
  (class cplal external)
  (doc "used to catch windowunderflow before the return has completed")
  (update
    (And (Not WAIT) (Memq OPCODE1 '(ret1 ret3 ret5 ret7))) )))

(defnode winunderflow
  (depends changedCWP phi3) ;SWPs changeCWP2t omitted
  (doc "Underflow occurs if the CWP is one less than the SWPs and a return is about to be made and the W option bit is set")
  (update
    (If3way phi3
      (If3way changeCWP2t
        (Equal changedCWP (Bits 6 4 SWPs))
        OFF
        UNK)
      winunderflow
      UNK)
    )
  )
)
SOFTWARE INTERRUPT DETECT LOGIC

(defnode SWI
(depends phi2) ; Alzeroforce CPIPE1s PSW omitted
(doc "software interrupt on a call or jmp (fast shuffle instruction)
    depends on bit 6 of the CPIPE (normally the % bit")
(update
(If3way phi2
    (And Alzeroforce
        (numbtosymb (Bits 0 0 PSW))
        (numbtosymb (Bits 6 6 CPIPE1s)))
    SWI
    UNK
    )
)

; TRAP/INTERRUPT signal generation logic
;
; NOTE: all tag trap conditions except for Mem(RX+ S2) and overflow/under
;       flow are valid at the end
;       of phi2.
;       No traps or interrupts allowed on power up
;
(defnode TAGtrap
(depends tCPIPE1s notanINT loadTRAP RXint) ; tagged mode?
(class tpla internal external)
(update
(And (numbtosymb (Bits 6 6 tCPIPE1s)) ; tagged mode?
    (Or notanINT loadTRAP RXint)
    )
)
)

(defnode TRAP
(depends phi3 phi1 RESET WAIT2 validtrapi GStrap latetrap intTAGtrap
    illegalopc SWI winoverflow winunderflow instrpagefINT IOINT datapagefINT)
(update
(If3way RESET
OFF
(If3way phi2
TRAP
(If3way (And phi1 (Not WAIT2))
(And (Not latetrap)
(Or validtrap
  GStrap
  intTAGtrap
  illegalopc
  SWI
  winoverflow
  winunderflow
  instrpagefINT
  IOINT
  datapagefINT))
  TRAP
  UNK)
  UNK)
  UNK)))
(defnode latetrap
  (depends phi2)
  (update
    (If3way phi2
     TRAP
     latetrap
     UNK)))
(defnode CPIPEtrap
  (depends latetrap TRAP)
  (update
    (Not (Or latetrap (Not TRAP)))))

;***********************************************************************

; Trap/Interrupt Priority Encoder Logic
;
; Crude priority encoder (Vector assignments as per table
;  6.1 of SOAR architecture document).
;
; The trap pla2 is used for encoding of the trap reason. there
; may be a more efficient way but since the necessary signals
; are already inputs to the PLA, this method was chosen.
; The encoding was only slightly optimized, in the interest of
; minimizing errors. It is assumed POP will do the rest.

(defnode trapinstr
(defnode validtrap1
  (depends pCONDvalid trapinstr)
  (update
   (And pCONDvalid trapinstr)))

(defnode GStrap
  (depends S1older nonLIFO RDcontext)
  (class tpla external)
  (doc "indicates when a Garbage collection/scavenging trap is executing,
         intended to make the trap reason encoding easier to read")
  (update
   (And (Or S1older nonLIFO RDcontext) (numbtosymb (Bits 6 6 tCPIPEls))))
)

(defnode intTAGtrap ;internal TAG trap
  (depends TAGtrap aluVout pov_unflow)
  (doc "logically the same as externally generated TAGtrap signal, except
         without the skip disqualifier. Not a critical path")
  (update
   (Or TAGtrap (And aluVout pov_unflow))
   )
)

;*****************************************************************

(defnode TRAPreason0
  (depends IOINT instrpagefINT GStrap validtrap1 datapagefINT winunderflow
       winoverflow SWI intTAGtrap illegalopc)
  (class tpla2 external)
  (update
   (And (Not illegalopc)
     (Or (And IOINT
          (Not instrpagefINT)(Not validtrap1)(Not winunderflow)
          (Not SWI))
     (And GStrap
      (Not validtrap1)(Not winunderflow)(Not SWI))
     (And datapagefINT
      (Not winunderflow)(Not SWI))
  )
  )
(And winoverflow
   (Not SWI))
intTAGtrap))
)
)

(defnode TRAPreason1
   (depends GStrap validtrapi datapagefINT winunderflow winoverflow SWI
   intTAGtrap illegalopc)
(class tpla2 external)
(update
   (And (Not (Or illegalopc intTAGtrap))
     (Or (And GStrap
       (Not datapagefINT)(Not winunderflow))
     validate
       (Not datapagefINT)(Not winunderflow))
     winoverflow
SWI))
)
)

(defnode TRAPreason2
   (depends GStrap validtrapi datapagefINT winunderflow winoverflow SWI
   intTAGtrap illegalopc)
(class tpla2 external)
(update
   (And (Not (Or winoverflow SWI intTAGtrap illegalopc)).
     (Or GStrap
       validtrapi
       datapagefINT
       winunderflow))
   )
)

(defnode TRAPreason3
   (depends IOINT instrpagefINT GStrap validtrapi datapagefINT winunderflow
   winoverflow SWI intTAGtrap illegalopc)
(class tpla2 external)
(update
 (And (Not (Or GStrap validtrapi datapagefINT winunderflow winoverflow
   SWI intTAGtrap illegalopc))
 (Or instrpagefINT IOINT))
 )
)

;******************************************************************
(defnode TRAPreason
    (depends phi1 phi3)
    (doc "this node collects the four bits out of tpla that form the trap reason into a single value")
    (update
      (If3way phi3
        TRAPreason ;refresh
        (If3way (And phi1 (Not WAIT2))
          (Logor (symbtonumber TRAPreason0)
            (Logor (Lsh (symbtonumber TRAPreason1) 1)
              (Logor (Lsh (symbtonumber TRAPreason2) 2)
                (Lsh (symbtonumber TRAPreason3) 3)))))
        TRAPreason
        UNK
      )
    )
    UNK
  )
)
EXTERNAL LOGIC/SIGNALS

Description of external inputs so that slang description can be
simulated. This is a temporary kluge.

these nodes are made arbitrarily dependent on one clock phase so
that they and all their descendents are evaluated only once per
cycle

(defnode RESETin
  (depends phi3)
  (update RESETin)
)

(defnode WAITin
  (depends phi3)
  (update WAITin)
)

(defnode pageINTin
  (depends RESETin)
  (update OFF))

(defnode IOINTin
  (depends RESETin)
  (update OFF)
)

; The following nodes model the external fast shuffle logic

(defnode MALlatch
  (depends phi1)
  (update
    (If3way phi1
      MALm
      MALlatch
      UNK))
)

(defnode extaddrmux
(depends FSHCNTL MAL latch extMALs)
(doc "the external address mux that selects between the
addr output by the chip and the addr latched in an
external latch on the previous instruction fetch.")
(update
 (If3way FSHCNTL
   (Bits 27 0 MAL latch) ; take the memory addr output from the chip
   extMALs ; take the external memory addr latch value
   UNK
   )
)

; NOTE: For this extMAL to function properly, traps and interrupts must
; restart the instruction just fetched.

(defnode extMALm
 (depends phi3 DATABUSin WAIT) ; I_D omitted
 (doc "the external memory addr latch")
 (update
 (If3way (And phi3 (Not WAIT) I_D) ; only latch if an instruction fetch
   (Bits 27 0 DATABUSin) ; get lower 28 bits
   extMALm ; refresh is not explicitly included here
   UNK
   )
)
)

(defnode extMALs
 (depends phi1) ; extMALm omitted
 (update
 (If3way phi1
   extMALm
   extMALs
   UNK
   )
)
)
MEMORY

The off-chip memory is represented in the form of an association-list which is the value of the (global) variable: memory_list

The pairs in the above association list consist of:
- A 28 bit integer address (in octal)
- A list which represents symbolically the contents of that word

This list can be:
- An 8 field instruction
- A 2 field integer

The formats of these lists are:

\( (s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8) \)

- \( s_1 \): i for instruction
- \( s_2 \): the integer representing the assembled value
- \( s_3 \): % for non-tagged (0), n% for tagged (1) for calls and jumps
  - n% means SI on (1)
  - % means SI off (0)
- \( s_4 \): symbolic opcode (ex: reti, add)
- \( s_5 \): dst reg # (ex: 0, 17, 31) (data src reg for stores)
- \( s_6 \): src1 reg # (drives busA)
- \( s_7 \): i for immediate, ni for not immediate (must be i for stores)
- \( s_8 \): src2 reg or immediate data
  - immediate data in octal (ex: #o0351)

\( (s_1, s_2) \)

- \( s_1 \): d for data
- \( s_2 \): data (in octal)

; Symbolic to numeric translation

(defun memory_s2n (s)
  (let ((type (car s)))
    (cond
      ((eq type 'i)
       (cadr s))
      )))
((eq type 'd)
  (If (eq I_D ON)
    then
      (warning "attempting to fetch data as an instruction")
      ; We do not need/want more than the warning message.
      ; Otherwise, slang will halt before the last instruction
      ; has a chance to finish! This causes problems when
      ; comparing outputs of simulators. It would also cause a
      ; problem if there is a data word following a return
      ; instruction: the data word is fetched but not executed.
      ; Ideally, we want to stop if a data word enters CPIPE1s, but
      ; that would require more logic and code than I am willing
      ; to do right now.
      ; If you want slang to stop without human intervention, the
      ; only way to do it is by counting clocks.
    )
  )
  (If (equal (type (cadr s)) 'bignum)
    then
      (iofb (cadr s))
    else (cadr s))))

; all this goes away
; (If (or (eq (caddr s) 'call)
;       (eq (caddr s) 'jmp))
;    then (Logor
;      (If (not (eq (cadr s) '%))
;        then (Lsh 1 29) ; from bit 0 to bit 29
;        else 0)
;      (Logor (Lsh (Bits 5 0 (symbtonumopcode(caddr s))) 23)
;          ; opcode
;          (caddrr s))) ; address
;    else (Logor
;      (Lsh 1 30) ; not a fast shuffle instruction
;    (Logor
;      (If (not (eq (cadr s) '%))
;        then (Lsh 1 29) ; from bit 0 to bit 29
;        else 0)
;      (Logor (Lsh (Bits 5 0 (symbtonumopcode (caddr s))) 23)
;          (+ (If (or (eq (caddr s) 'store)
;               (eq (caddr s) 'storem))
;              then (Lsh (Bits 11 7 (caddddddr s)) 18)
;             ; upper 5 bits of constant goe
;             into normal DST field area
;             else (Lsh (caddrr s) 18)) ; normal DST
;          (Lsh (caddddr s) 13) ; SRCl
;          (If (eq (cadddddr s) 'i)
;            then (Lsh 1 12) ; set immed on
;;
;; else 0)
;; (If (or (eq (caddr s) 'store)
;; (eq (caddr s) 'storem))
;; then (+ (Lsh (cadddr s) 7)
;; (Bits 6 0 (cadddddr s)))
;; else (If (eq (cadddddr s) 'i)
;; then (Bits 11 0 (cadddddddдр s))
;; ; beware them negative immeds!
;; else (Lsh (cadddddddдр s) 7)))))

((eq type 'd)
 (If (eq L_D ON)
 then
 (warning "attempting to fetch data as an instruction")
 (simulationend)
 else
 (If (equal (type (cadr s)) 'bignum)
 then
 (iofb (cadr s))
 else (cadr s)))
 (t UNK)))

;*******************************************************************
; Function to convert bignum to fixnum, courtesy of keith sklower

(defun iofb (arg)
 (prog (handy)
 (setq handy (bignum-to-lbt arg))
 (setq handy (+ (car handy) (bh (cadr handy) 30)))
 (return handy))

;*******************************************************************
;
; Symbolic to Numeric opcode correspondence
;
; Note: This opcode assignment is tentative (3/23/83)
; 30 "opcodes" encoded in 7 bits. MSB bit distinguishes fast
; shuffle instructions
;
;*******************************************************************

(declare (special symbopcenc))

(setq symbopcenc '( (flush #o104) (srl #o140)
 (TRAP #o105) (sll #o151)
 (SKIP #o106) (sra #o142)
 (ret0 #o110) (xor #o144))

*******************************************************************
(defun symbtonumopcode(s)  ; opcode is given in octal
   (let ((x (assq s symbopcenc)))
      (if x then (cadr x) else (if (numberp s) then s else UNK))))

; NOTE: if x is non-nil, then if evaluates
;   loadc is the load class instruction
;   jmp really occupies opcode space o4x, o5x, o6x, o7x
;   call really occupies opcode space o0x, o1x, o2x, o3x
;   Call, flush, and exception are not user visible.
;   Flush is forced into the pipe for flush a currently executing
;   instruction (on ret, or traps)

;**************************************************************************
; Numeric to Symbolic translation
(defun memory_n2s (n)
  (list 'd n))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;------ MEMORY_INITIALIZE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(declare (special memory_list prog_files))

(defun memory_initialize ()
  ; Load the files given in prog_files (in sim.l)
  (setq memory_list (list '(#x10000000 (d 0)))) ; dummy
  (do ((files prog_files (cdr files)))
    ((null files))
    (load (uconcat 'progs/ (car files))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;------ MEMORY_READ
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun memory_read (addr)
  ; Return the numeric contents of that location
  (let ((pair (assoc addr memory_list)))
    (If (null pair)
      then 'UNK ; uninitialized
      else (memory_s2n (cadr pair)))) ; contents as numeric data!

(defun MemoryContent (Addr)
  ;
  ; Used by memorycheck, defined by larus.
  ;
  (memory_read Addr))

(defun smemread (addr)
  ; Return the symbolic contents of that location
  (let ((pair (assoc addr memory_list)))
    (If (null pair)
      then 'UNK ; uninitialized
      else (cadr pair)))) ; as symbolic data!

(defun nummemprint ()
  (let ((cntr 0))
    (do ((port (outfile 'instrfile))
      ......
(inst (assoc cntr memory_list)))
((null inst) (close port))
(cprintf '|%x| (memory_s2n (cadr inst)) port)
(terpr port)
(setq cntr (1+ cntr))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;; MEMORY_PRINT

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun memprint () ; print the contents of the memory
  (let ((addr (- (caar memory_list) 1)))
    (do ((lbt memory_list (cdr lbt)))
        ((null lbt))
      (let* ((pair (car lbt))
               (cont (cadr pair))
               (type (car cont)))
        (If (not (= & (car pair) (1+ addr)))
            then (terpri))
        (setq addr (car pair))
        (If (memq type '(d))
            then (If (bigp (cadr cont))
                    then (setq cont
                        (ConvertToPrintBase (iofb (cadr cont))))
                    else (setq cont (ConvertToPrintBase (cadr cont)))))
        (else (setq type 'i))
        (princ addr) (princ 'j |)
        (princ type) (princ '| j)
        (princ cont) (terpri))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; ; RFILE_PRINT

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun rfprint (cwp)
  ; Print the window of 8 registers pointed to by the cwp parameter
  (let ((startcnt)
          (stopcnt))
    (If (eq cwp 'specials)
        then (setq startcnt 0)
        (setq stopcnt 0)
        (setq cwp 0) ; CWP is immaterial
        (printspecials)
        elseif (eq cwp 'globals)
then
(setq startcnt 31)
(setq stopcnt 23)
(setq cwp 0)  ; CWP is immaterial
elseif (eq cwp 'highs)
    then
        (setq startcnt 15)
        (setq stopcnt 7)
        (setq cwp CWPs)
    elseif (eq cwp 'lows)
        then
            (setq startcnt 7)
            (setq stopcnt -1)
            (setq cwp CWPs)
        else
            (setq startcnt 15)
            (setq stopcnt 7)
    (do ((i startcnt (- i 1))
          (temp))
          ((=& i stopcnt))
          (princ '|Register |)
          (princ i)
          (princ '| |)
          (setq temp (ConvertToPrintBase (rf (regdecode cwp i))))
          (If (equal temp UNK)
              then (princ 'UNK)
              else (princ temp))
          (terpri))))

;********************************************************************

(defun printspecials ()
(let (())
    (princ '|OPC,PSW,shDST |)
    (princ
        (ConvertToPrintBase (Logor (Lsh PSW 5) (Logor (Lsh shOPC 8) shDST)))))
    (terpri)
    (princ '|CWP |)
    (princ (ConvertToPrintBase CWPs))(terpri)
    (princ '|TB |)
    (princ (ConvertToPrintBase TBs))(terpri)
    (princ '|SWP |)
    (princ (ConvertToPrintBase SWPs))(terpri)
    (princ '|SHA |)
    (princ (ConvertToPrintBase SHA))(terpri)
    (princ '|SHB |)
    (princ (ConvertToPrintBase SHB))(terpri)
    (princ '|PC |)
    (princ (ConvertToPrintBase PCs))(terpri)
))

;********************************************************************
(defun RegisterContent (Number)
  ;
  ; Used by registercheck, defined by larus, Pendleton.
  ;
  (if (eq Number 23)
      then (Logor (Lsh PSW 5) (Logor (Lsh shOPC 8) shDST))
    elseif (eq Number 22)
      then (Lsh CWPs 4)
    elseif (eq Number 21)
      then TBs
    elseif (eq Number 20)
      then SWPs
    elseif (eq Number 19)
      then SHA
    elseif (eq Number 18)
      then SHB
    elseif (eq Number 17)
      then PCs
    elseif (eq Number 16)
      then 0
    else (rf (regdecode CWPs Number))))

;---------------------------------------------------------------------

;------ MEMORY_WRITE
;---------------------------------------------------------------------

(defun memory_write (addr data)
  ; Returns t if successful, and nil iff (unsuccessful) attempt to
  ; write instruc.
  (let ((pair (assoc addr memory_list)))
    (if (null pair) ; not there yet
      then
        (nconc memory_list ; put it
          (list (setq pair (list addr (lbt 'd 0)))))))
    ; DETERMINE CONTENTS
    (if (eq (car (cadr pair)) 'i)
      then nil
    ; WRITE
    (rplaca (cdr pair) (memory_n2s data))
    t))

;**********************************************************************

(defun memorywrite
  (depends phi3 ) ;RD_WR, DATABUSin extaddrmux omitted
(update
  (If3way (And phi3 (Not RD_WR))
    (If (not (memory_write extaddrmux DATABUSin))
      then (warning "self modifying code!!"))
    memorywrite
    UNK
  )))
Appendix C
Circuit Block Logic Diagrams

Block Diagram

Datapath-Dataflow

DatOut  SXT/DIL  DIL

LOADL  InputDr

Register File

Precharge

SHA  SHB

DestDr  INAm

INBm  EX/INS

PrechargeD

ALU

ALUDr  Destlatch

firstPC  PCIncr

lastPC

TB  MAL

SWP  SWPcompare
Datapath-Control Lines

- Arrows with only one marked terminal come from or go to control.

Block Diagram (cont.)
DataOut

\[ \begin{array}{ccc}
\text{busL} & \text{rd/wr} & \text{Vdd} \\
& & 2.4 \Omega/4 \\
& & 30/4 \\
& & \text{Vdd} \\
& & 2.6 \Omega/4 \\
& & 3.2 \text{pf} \\
& & \text{dbin} < n > \\
\end{array} \]

DataOut cell
SXT/DIL

storeSXT
storeSXT*
sign extender

8 SXT1 cells
2 SXT1 cells

32 SXT2 cells

7 SXT1 cells
SXT/DIL (cont.)

SXT1 cell

SXT2 cell
DIL

32 DIL cells

DIL cell

CPIPEload#1  phi1#0  phi2#1  passSXT

dbin<n>  

busL<n>

dbin<n>  

busL<n>
LOADL

32 LOADL cells

DATABUS into LOADL

WAIT\#0\, WAIT\#\, phi1\#1, LOADL to busL

LOADL<\n\>

dbin<\n\>, busL<\n\>, busB<\n\>

LOADL cell
InputDr

32 InputDr cells

LOADLtobusA Vdd
LOADLtobusB Vdd

LOADL<cell>

busA<n>
busB<n>

InputDr cell
Register File

72x32 register cells

32 register cells

72 register cells

RFtoA<\text{j,k}> \quad \text{RFtoB<\text{j,k>}}

busA<\text{n}> \quad \text{busB<\text{n>}}

register cell
Precharge

32 Precharge cells

Precharge cell
SHA

32 SHA cells

SHA cell
SHB

32 SHB cells

SHB cell
DestDr

\[
\text{32 DestDr cells}
\]

\[
\text{DestDr cell}
\]
16 StoAblank cells

8 StoA cells

StoAblank cell

3 StoA0 cells

4 StoA cells

busStobusA

busS<\text{n}>

busA<\text{n}>

as shown above

StoA0 cell
StoA (cont.)

As shown above

StoA blank cell

As shown above

StoA cell
INAm

\[
\begin{align*}
32 \text{ INAm cells}
\end{align*}
\]

\[
\text{busDtoINA} \quad \text{busAtoINA} \quad \text{phi3}\#0
\]

INAm cell
INBm

32 INBm cells

INBm cell
EX/INS (cont.)

32 EX/INS cells

EX/INS pass

INAm<n> ----------------- Ain<n>
INAm*<n> ----------------- Ain*<n>
INBm<n> ----------------- Bin<n>
INBm*<n> ----------------- Bin*<n>

EX/INS cell
PrechargeD

32 PrechargeD cells

PrechargeD cell
## ALU

<table>
<thead>
<tr>
<th>nibble_MSB</th>
<th>nibble</th>
<th>nibble</th>
<th>nibble</th>
<th>nibble</th>
<th>nibble</th>
<th>nibble</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>bussel_31</th>
<th>funcs</th>
<th>shft_31</th>
<th>sum</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bussel_30</td>
<td>funcs_30</td>
<td>shft_30</td>
<td>sum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bussel</td>
<td>funcs</td>
<td>shft</td>
<td>sum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bussel</td>
<td>funcs</td>
<td>shft</td>
<td>sum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bussel</td>
<td>funcs</td>
<td>shft</td>
<td>sum</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nibble_MSB</th>
<th>nibble</th>
<th>nibble</th>
<th>nibble</th>
<th>nibble</th>
<th>nibble</th>
<th>nibble</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>bussel</th>
<th>funcs</th>
<th>shft</th>
<th>sum</th>
<th>carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>bussel</td>
<td>funcs</td>
<td>shft</td>
<td>sum</td>
<td></td>
</tr>
<tr>
<td>bussel</td>
<td>funcs</td>
<td>shft</td>
<td>sum</td>
<td></td>
</tr>
<tr>
<td>bussel</td>
<td>funcs</td>
<td>shft</td>
<td>sum</td>
<td></td>
</tr>
<tr>
<td>bussel</td>
<td>funcs</td>
<td>shft</td>
<td>sum</td>
<td></td>
</tr>
</tbody>
</table>
ALU (cont.)

```
 selB\bar
 A_1{}^{*<31>}
 Bin^{<31>}

 Bin^{*<31>}
 bussel_31

 selB\bar
 A_1{}^{*<30>}
 Bin^{<30>}

 Bin^{*<30>}
 bussel_30

 selB\bar
 Bin^{<n>}

 Bin^{*<n>}
 bussel
```
ALU (cont.)

funcs_30

funcs
ALU (cont.)

\[\text{Ain}^{\nu13}, \text{ALU}^{\nu13}, \text{shft}_{\nu31}^{\nu}, \text{pass}_{\nu13}^{\nu}, \text{sel}_{\nu13}^{\nu}, \text{SR}\]

\[\text{Ain}^{\nu13}, \text{ALU}^{\nu13}, \text{shft}_{\nu30}^{\nu}, \text{pass}_{\nu13}^{\nu}, \text{sel}_{\nu13}^{\nu}, \text{SR}\]

\[\text{Ain}^{\nu}, \text{ALU}^{\nu}, \text{shft}_{\nu}^{\nu}, \text{pass}_{\nu}^{\nu}, \text{sel}_{\nu}^{\nu}, \text{SR}\]
ALU (cont.)

\[ \begin{align*}
\text{selaluAND} & \quad \text{selaluXOR} & \quad \text{selaluSUM} \\
\text{xnor} & \quad \text{and} & \quad \text{xor} \\
\text{ALU} & \quad \text{sum} & \quad \text{Cin*} \\
\end{align*} \]

\[ \begin{align*}
V_{dd} & \quad \text{4/4} \\
\text{D1} & \quad \text{16/4} \\
\end{align*} \]
ALU (cont.)

carry
ALUDr

30 ALUDr cells

ALU<31> -> EAbus<31> -> ALU31

ALU<30> -> EAbus<30> -> ALU30

ALU<n> -> EAbus<n> -> ALUDr cell
Destlatch (cont.)

Destlatch1 cell

Destlatch2 cell
firstPC (cont.)

**firstPC1 cell**

**firstPCblank cell**
PCIncr

\[
\begin{array}{c}
16 \text{ PCIncrPair cells} \\
\end{array}
\]

PCIncrPair cell

PCIncrEven cell

PCIncrOdd cell
PCIncr (cont.)

PCIncrOdd cell

PCIncrEven cell

PCIncr PCin*<n+1> Vdd

PCIncr PCIn<n>
lastPC

32 lastPC cells

lastPC cell
22 TB cells

10 TB blank cells

write to TB

phi1#8  phi2#5  read TB to A

busD<n>  

TB cell
TB (cont.)

readTBtoA

busD<n> busD<n>

TBblank cell
MAL

32 MAL cells

MAL cell

as shown above

EAbus<

PCm<

phi2#7

addr* <n>
SWP

32 SWP cells

SWP cell
SWP compare

Vdd

PTRtoREG

PTRtoREG*

CMP drive cell

Sin<4>
SWPcompare (cont.)

CMPoddDC cell

CWPevenDC cell
SWPcompare (cont.)

CMPoddOR cell

CMPevenOR cell
AddressOut

32 AddressOut cells

AddressOut cell
Decoders

<table>
<thead>
<tr>
<th>S2window</th>
<th>D2window</th>
<th>S1window</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ddecode</td>
<td>PTRmux</td>
<td></td>
</tr>
<tr>
<td>S2decode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1decode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>muxes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

decodeEA\*  
decodeEA\*  
DST2s\(<n>\)  
DSTs\(<n>\)  
DST2s\(<n>\)*  
DSTs\(<n>\)*  
pdecodeEA\*  
pdecodeEA\*  
SRC2m1\(<n>\)  
DSTM\(<n>\)  
SRC2m1\(<n>\)*  
DSTM\(<n>\)*  
SRC2\(<n>\)  
DST2\(<n>\)*  
DST2\(<n>\)*  

PTRmux
Decoders - S2window

SRC2<4>

CWPm<2>

CWPm<0>

SRC2<3>*

SRC2<4>*
Decoders- Ddecode

D2cwp \langle j \rangle

DST2 \langle 2 \rangle

DST2 \langle 2 \rangle^\ast

DST2 \langle 1 \rangle

DST2 \langle 1 \rangle^\ast

DST2 \langle 0 \rangle

DST2 \langle 0 \rangle^\ast

\text{nonnullreturn} \#0

DST2 \text{inputs}

V_{dd}

D2cwp \langle j \rangle
Decoders- S2decode

\[
\begin{align*}
S2\text{cwp} & \leftarrow \text{dec2} \leftarrow \text{dec2} \leftarrow \text{dec2} \leftarrow \text{dec2} \leftarrow \text{dec2} \leftarrow \text{dec2} \leftarrow \text{dec2} \\
\text{SRC2} \leftarrow 2 & \leftarrow \text{dec2} \leftarrow \text{dec2} \leftarrow \text{dec2} \\
\text{SRC2} \leftarrow 2 & \leftarrow \text{dec2} \leftarrow \text{dec2} \\
\text{SRC2} \leftarrow 1 & \leftarrow \text{dec2} \leftarrow \text{dec2} \\
\text{SRC2} \leftarrow 1 & \leftarrow \text{dec2} \leftarrow \text{dec2} \\
\text{SRC2} \leftarrow 0 & \leftarrow \text{dec2} \leftarrow \text{dec2} \\
\text{SRC2} \leftarrow 0 & \leftarrow \text{dec2} 
\end{align*}
\]
Decoders - S1decode

S1cwp<1>

SRC1m1<2>

SRC1m1<2>*

SRC1m1<1>

SRC1m1<1>*

SRC1m1<0>

SRC1m1<0>*

...
Decoders-muxes

- readA
- dec1\(<j, k>\)
- write
- phi3*
- dec3\(<j, k>\)
- readB
- phi2*
- dec2\(<j, k>\)

\[ \text{D2} \]

\[ \text{T} \]
PSW (cont.)

PSW<0> cell
shOPC

shOPCDr

8 shOPC cells

PSW<1>\* 

TRA 

ppbusSHADOW\* 

phi2* 

phi1* 

preadPSWtoA* 

phi2* 

busSHADOW#1 

phi1#9 

readPSWtoA#1 

shOPCDr

busSHADOW#1 

phi1#9 

readPSWtoA#1

CPIPE1s <n> 

shOPC<n> 

busS<n+8> 

busS<n+8> 

shOPC cell
shDST

\[ \text{write to PSW1} \rightarrow \text{write to PSW0} \]
\[ \text{phi3} \]
\[ \text{PSW} \rightarrow \text{bus SHADOW0} \]
\[ \text{phi2} \]
\[ \text{phi1} \]
\[ \text{read PSW to A} \rightarrow \text{phi1#10} \]
\[ \text{phi2} \]
\[ \text{shDSTDr} \]

shDST

\[ \text{bus SHADOW1} \rightarrow \text{bus SHADOW0} \]
\[ \text{phi2} \]
\[ \text{phi1} \]
\[ \text{read PSW to A} \rightarrow \text{phi1#10} \]
\[ \text{phi2} \]
\[ \text{shDST cell} \]

\[ \text{bus S} \rightarrow \text{bus S} \]
SRC2 (cont.)

 SRC2cl

 SRC2decr
DST1 (cont.)

DST1cl

DST1decr
DST2 (cont.)

DST210 cell

DST211 cell
**CPIPE1**

<table>
<thead>
<tr>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>cPlm</td>
<td>cPlF</td>
<td>cPlT</td>
<td>cPlS</td>
<td>cPlL</td>
<td>cPlR</td>
<td>cPlM</td>
</tr>
</tbody>
</table>

**CPIPE1Dr**

- **RESET**
  - phi3
  - CPIPE1load1
  - CPIPE1load1
  - phi3
  - phi2
  - CPIPE1step
  - phi3
  - CPIPE1flush
  - phi3
  - CPIPE1trap
  - CPIPE1skip
  - phi3
  - CPIPE1loadc
  - phi3
  - CPIPE1store
  - phi3
  - CPIPE1loadm
  - phi3
  - CPIPE1storem
  - phi3
  - WAIT2
  - phi1
  - CPIPE1Dr

- **RESET #2**
  - phi2
  - #15
  - #3
  - #5

- **CPipe1load #5**

- **phi2 #15**

- **CPIPE1step #3**

- **CPIPE1flush**

- **skip**

- **CPIPE1loadc**

- **CPIPE1store**

- **CPIPE1loadm**

- **CPIPE1storem**

- **WAIT #8**
CPIPE1- CP1s

10 CP1s cells

CP1m \( \langle n \rangle \) \quad \text{WAIT}\#6 \quad \text{phi2}\#15 \quad \text{CP1s cell}

CPIPE1s \( \langle n \rangle \)
CPIPE1- CP1F

CPIPE1flush

<table>
<thead>
<tr>
<th></th>
<th>CP1m&lt;9&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vdd</td>
<td>CP1m&lt;8&gt;</td>
</tr>
<tr>
<td></td>
<td>CP1m&lt;7&gt;</td>
</tr>
<tr>
<td></td>
<td>CP1m&lt;6&gt;</td>
</tr>
<tr>
<td></td>
<td>CP1m&lt;5&gt;</td>
</tr>
<tr>
<td></td>
<td>CP1m&lt;4&gt;</td>
</tr>
<tr>
<td></td>
<td>CP1m&lt;3&gt;</td>
</tr>
<tr>
<td>Vdd</td>
<td>CP1m&lt;2&gt;</td>
</tr>
<tr>
<td></td>
<td>CP1m&lt;1&gt;</td>
</tr>
<tr>
<td></td>
<td>CP1m&lt;0&gt;</td>
</tr>
</tbody>
</table>
CPIPE1 - CP1T

CPIPEtrap

CP1m<9>

CP1m<8>

CP1m<7>

CP1m<6>

CP1m<5>

CP1m<4>

CP1m<3>

CP1m<2>

CP1m<1>

CP1m<0>

Vdd
CPIPE1- CP1S

![Diagram of CPIPE1- CP1S](image-url)
CPIPE1 - CP1L

CPIPE1loadc

\begin{verbatim}
CP1m<9>
CP1m<8>
CP1m<7>
CP1m<6>
CP1m<5>
CP1m<4>
CP1m<3>
CP1m<2>
CP1m<1>
CP1m<0>
\end{verbatim}
CPIPE1- CP1R

CPIPE1store

\[ \text{Vdd} \quad \text{CP1m}^{<9>} \]
\[ \text{Vdd} \quad \text{CP1m}^{<8>} \]
\[ \text{Vdd} \quad \text{CP1m}^{<7>} \]
\[ \text{Vdd} \quad \text{CP1m}^{<6>} \]
\[ \text{Vdd} \quad \text{CP1m}^{<5>} \]
\[ \text{Vdd} \quad \text{CP1m}^{<4>} \]
\[ \text{Vdd} \quad \text{CP1m}^{<3>} \]
\[ \text{Vdd} \quad \text{CP1m}^{<2>} \]
\[ \text{Vdd} \quad \text{CP1m}^{<1>} \]
\[ \text{Vdd} \quad \text{CP1m}^{<0>} \]
CPIPE1- CP1M

CPIPE1loadm

- CP1m<9>
- CP1m<8>
- CP1m<7>
- CP1m<6>
- CP1m<5>
- CP1m<4>
- CP1m<3>
- CP1m<2>
- CP1m<1>
- CP1m<0>
CPIPE1 - CP1N

CPIPE1storem

- CP1m<9>
- CP1m<8>
Vdd - CP1m<7>
- CP1m<6>
Vdd - CP1m<5>
Vdd - CP1m<4>
Vdd - CP1m<3>
SRC2sub<2> - CP1m<2>
SRC2sub<1> - CP1m<1>
SRC2sub<0> - CP1m<0>
CPIPE1- CP1m

dbin<31> CPm0 CP1m<9>
dbin<12> CPm0 CP1m<8>
dbin<30> CPm1 CP1m<7>
dbin<29> CPm0 CP1m<6>
dbin<28> CPm0 CP1m<5>
dbin<27> CPm0 CP1m<4>
dbin<26> CPm0 CP1m<3>
dbin<25> CPm1 CP1m<2>
dbin<24> CPm0 CP1m<1>
dbin<23> CPm0 CP1m<0>

RESET#2 CPIPE1load#5 phi2#15 CPIPE1step#3

As shown above

dbin<k> CPm0 CP1m<n> as shown above

As shown above

RESET#2 CPIPE1load#5 phi2#15 CPIPE1step#3

As shown above

dbin<k> CPm1 CP1m<n> as shown above

As shown above
CPIPE2

CPIPE2Dr

CPIPE1s<7> --- CPIPE21 cell
CPIPE1s<6> --- CPIPE21 cell
CPIPE1s<5> --- CPIPE20 cell
CPIPE1s<4> --- CPIPE20 cell
CPIPE1s<3> --- CPIPE20 cell
CPIPE1s<2> --- CPIPE21 cell
CPIPE1s<1> --- CPIPE20 cell
CPIPE1s<0> --- CPIPE20 cell

CPIPE2Dr

WAIT

\phi_3

\phi_1\#12

\phi_2\#14
CPIPE2 (cont.)

CPIPE21 cell

CPIPE20 cell
CWP (cont.)

CWP cell
**DSTs**

- **DSTsDr**
  - 7 DSTs cells

- **DSTsDr**
  - phil* → phi1
  - phi2* → phi2

**DSTs cell**

- DSTm<n>* → phi1#13
- DSTm<n>* → phi2#15
Driver2

readRF_accessA1*   \rightarrow  readA
phi2*              \rightarrow  readB
readRF_accessB1*   \rightarrow  readB
phi2*              \rightarrow  nillonreturn
nillonreturn        \rightarrow  nillonreturn\#0
writeRF_access2    \rightarrow  write
STOREwrite         \rightarrow  write
phi3*              \rightarrow  write
Driver3

\begin{align*}
\text{phi1}* & \quad \text{PrechargeAB} \\
\text{AiZero1}* & \quad \text{AiZero} \\
\text{phi2}* & \quad \text{PrechargeL} \\
\text{PSW} < 1 > * & \quad \text{busAtoSHA} \\
\text{TRAP} & \quad \text{busLtoSHB} \\
\text{bus} & \quad \text{busBtoSHB} \\
\text{write} & \quad \text{phi1#2} \\
\text{phi3}* & \quad \text{SHAtobusA} \\
\text{phi1}* & \quad \text{busBtoINB} \\
\text{phi2}* & \quad \text{phi1#3} \\
\text{phi3}* & \quad \text{SHBtobusA} \\
\text{phi1}* & \quad \text{busDtobusAA} \\
\text{phi2}* & \quad \text{busDtobusA2} \\
\text{phi3}* & \quad \text{busDtobusB} \\
\text{phi2}* & \quad \text{phi1}* \\
\end{align*}
Driver 4

pbustobusA

phi2

pbusDtoINA

phi2

phi2

phi3

phi2

phi2

busStobusA

busDtoINA

busDtoINA

busAtoINA

phi3#0

busLtoINB

busBtoINB

busBtoINB

pbusBtoINB
Driver8

PCtoMAL1* ➔ PCtoMAL
phi3* ➔ phi3#
phi2* ➔ phi2#7

pALUtoMAL* ➔ ALUtoMAL
phi3* ➔ phi3#
phi2* ➔ phi2#8

TRAP* ➔ TRAP #0
phi3* ➔ phi3#

writetoSWP* ➔ writetoSWP
phi3* ➔ phi3#
phi2* ➔ phi2#8

readSWPtoA* ➔ readSWPtoA
phi2* ➔ phi2#8
drivers (cont.)

![Diagram of circuit connections involving Vdd, 4/16, 4/4, 16/4, and 5.8 pf with labels D5, D6, and D7.]

557
drivers (cont.)

![Diagram of electronic circuit with labels and components](image)

- **Vdd**
- **60/4**
- **20/4**
- **528/4**
- **3.2pf**
- **504/4**
- **D8**
Miscellaneous Logic
Vdd

SRC1 = DST2

SRC2 = DST2
Miscellaneous Logic (cont.)

\[
\begin{align*}
\text{SRC2} & < 4 \\
\text{SRC2} & < 3 \\
\text{SRC2} & < 2 \\
\text{SRC2} & < 1 \\
\text{SRC2} & = 0 \\
\end{align*}
\]

\[
\begin{align*}
\phi_1^2 & \quad \text{aluCINbar} \\
\phi_2^2 & \quad \text{CIN}^* < 0 > \\
\text{CPIPE} & \quad \text{fsHCNTL}^* \text{ (pad)} \\
\text{skipCOND} & \quad \text{valid} \\
\text{RD_} & \quad \text{WR}^* \text{ (pad)} \\
\text{I_} & \quad \text{D}^* \text{ (pad)} \\
\phi_1^2 & \quad \phi_2^2 \\
\text{WAIT}^2 & \quad \text{WAITACK}^* \text{ (pad)} \\
\text{RESET}^* & \\
\end{align*}
\]
Miscellaneous Logic (cont.)

phi3* — \(\quad\)
changeCWP1
changeCWP2t

Vdd

SWP<4>
SWP<5>
SWP<6>
chCWP<0>
chCWP<1>
chCWP<2>

\(\text{winoverflow}\)
\(\text{winunderflow}\)
Miscellaneous Logic (cont.)

\[ \text{phi}1^* \rightarrow \text{instrpagefINT} \]
\[ \text{phi}2^* \rightarrow \text{datapagefINT} \]
\[ \text{phi}3^* \rightarrow \text{PAGE}^* \]
\[ \text{pbusDtoINA}^* \rightarrow \text{JOINT} \]
\[ \text{PSW} < 1^* \rightarrow \text{Vdd} \]
\[ \text{IO}^* \rightarrow \text{Vdd} \]
\[ \text{Alzerofoce}^* \rightarrow \text{SWI} \]
\[ \text{CPIPE}1s < 6^* \rightarrow \text{illegalopc} \]
\[ \text{PSW} < 0^* \rightarrow \text{illegalopc} \]
\[ \text{pillegalopc} \rightarrow \text{intTAGtrap}^* \]
\[ \text{aluVout} \rightarrow \text{intTAGtrap} \]
\[ \text{TAGtrap} \rightarrow \text{pov\_unflow} \]
Miscellaneous Logic (cont.)

phi2*

RESET*

WAIT2

phi1*

pCONDvalid*

trapinstr*

GStrap

intTAGtrap

ilegalopcc

SWI

winoverflow

winunderflow

JINT

instpageINT

datapageINT

pCONDvalid*

skipCONDenabed

CPIPEtrap

TRAP

CPIPEskip*

skipCONDvalid
Miscellaneous Logic (cont.)

\[
\begin{align*}
\text{phi}^2 & \quad \rightarrow \\
\text{WAIT}^2 & \quad \rightarrow \\
\text{phi}^1 & \quad \rightarrow \\
\text{TRAPreason} < 3 > & \quad \rightarrow \quad \text{Reason} < 3 > \\
\text{TRAPreason} < 2 > & \quad \rightarrow \quad \text{Reason} < 2 > \\
\text{TRAPreason} < 1 > & \quad \rightarrow \quad \text{Reason} < 1 > \\
\text{TRAPreason} < 0 > & \quad \rightarrow \quad \text{Reason} < 0 >
\end{align*}
\]
Miscellaneous Logic (cont.)

CPIPE1s<6>
phi3*
Ain<31>
Ain<30>
Bin<31>
Bin<30>
ALU31
ALU30
Ain0

DST1<4>
DST1<3>
DST1<2>
DST1<1>
DST1<0>
A=B

cAinMSB
cBinMSB
caluMSB
cAin0
cDST1<4>
cDST1<3>
cDST1<2>
cDST1<1>
cDST1<0>
caluZout
Miscellaneous Logic (cont.)

phi2*

phi3*

Vdd

tCPIPE1s<0>
tCPIPE1s<1>
tCPIPE1s<2>
tCPIPE1s<3>
tCPIPE1s<4>
tCPIPE1s<5>
tCPIPE1s<6>
tCPIPE1s<7>
tCPIPE1s<8>
tbusA<31>
tbusA<31>
tbusB<31>
tbusB<31>
tbusB<30>
tbusB<30>
tbusB<29>
tbusB<29>
tbusB<28>
tbusB<28>
tagcompare
tagcompare
<table>
<thead>
<tr>
<th>op2</th>
<th>CPIPE2s&lt;9&gt;</th>
<th>&lt;7&gt;</th>
<th>&lt;5&gt;</th>
<th>&lt;4&gt;</th>
<th>&lt;3&gt;</th>
<th>&lt;2&gt;</th>
<th>&lt;1&gt;</th>
<th>&lt;0&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>call</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>jmp</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>flush</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TRAP</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SKIP</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ret0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ret1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ret2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ret3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ret4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ret5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ret6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ret7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>skip</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>trap1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>trap2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>trap3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>trap4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>trap5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>trap6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>trap7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>store</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>storem</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>load</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>loadc</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>loadm</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>srl</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sra</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>xor</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>and</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>or</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>add</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sl</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>sub</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>extract</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>insert</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>load0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>load1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>load2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>load3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>load4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>load5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>load6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>load7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>store0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>store1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>store2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>store3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>store4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>store5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>store6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>store7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>top1</td>
<td>tCPIPEDs&lt;9&gt;</td>
<td>&lt;7&gt;</td>
<td>&lt;5&gt;</td>
<td>&lt;4&gt;</td>
<td>&lt;3&gt;</td>
<td>&lt;2&gt;</td>
<td>&lt;1&gt;</td>
<td>&lt;0&gt;</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>call</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>jmp</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>flush</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TRAP</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SKIP</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ret0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ret1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ret2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ret3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ret4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ret5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ret6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ret7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>skip</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>trap1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>trap2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>trap3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>trap4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>trap5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>trap6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>trap7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>store</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>storem</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>load</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>loadc</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>srl</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sra</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>xor</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>and</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>or</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>add</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sll</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>sub</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>extract</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>insert</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>load0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>load1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>load2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>load3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>load4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>load5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>load6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>load7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>store0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>store1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>store2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>store3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>store4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>store5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>store6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>store7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>d1</td>
<td>cDST1&lt;4&gt;</td>
<td>&lt;3&gt;</td>
<td>&lt;2&gt;</td>
<td>&lt;1&gt;</td>
<td>&lt;0&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>---------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cpl al

\[
\text{CPIPE1load1} \leftarrow \text{Not (And \ \text{RESET} \ \text{Not \ \text{WAIT}}) (\ \text{Not (op1=load0 load1 load2 load3 load4 load5 load6 load7 store0 store1 store2 store3 store4 store5 store6 store7)))}
\]

\[
\text{pDATABUSintoLOADL} \leftarrow \text{And (Not \ \text{WAIT}) (\ \text{Not (op1=store storem store1 store2 store3 store4 store5 store6 store7)})}
\]

\[
\text{pALUtoPC*} \leftarrow \text{Not (And (op1=ret0 ret1 ret2 ret3 ret4 ret5 ret6 ret7 call jmp) \ \text{RESET} \ \text{Not \ \text{WAIT}}))}
\]

\[
\text{pPCI incr} \leftarrow \text{And \ \text{RESET} \ \text{Not \ \text{WAIT}} (\ \text{op1=flush TRAP SKIP insert extract add sub sll skip trap1 trap2 trap3 trap4 trap5 trap6 trap7 or xor sra srl and load0 store0)}
\]

\[
\text{lastPCload} \leftarrow \text{Or \ \text{WAIT} (op1=TRAP)}
\]

\[
\text{PCtoMAL1*} \leftarrow \text{Not (And (Not \ \text{WAIT}) (op1=flush SKIP load0 store0 skip trap1 trap2 trap3 trap4 trap5 trap6 trap7 sra srl or xor add and sub sll extract insert))}
\]

\[
\text{pALUtoMAL*} \leftarrow \text{Not (And (Not \ \text{WAIT}) (op1=ret0 ret1 ret2 ret3 ret4 ret5 ret6 ret7 call jmp load loadc loadm store storem load1 load2 load3 load4 load5 load6 load7 store1 store2 store3 store4 store5 store6 store7))}
\]

\[
\text{enableINTS1*} \leftarrow \text{Not (op1=ret4 ret5 ret6 ret7)}
\]
SRC2smin1* <- Not (And (Not WAIT) (op1=storem store2 store3 store4 store5 store6 store7))

DST1min1* <- Not (And (Not WAIT) (op1=load1 load2 load3 load4 load5 load6 load7))

PCstufoncall1* <- Not (And (Not WAIT) (op1=call) (Not (op1=TRAP)))

DST2step1* <- Not (And (Not WAIT) (Not (op1=call TRAP)))

CPIPE1flush* <- Not (And RESET* (op1=ret0 ret1 ret2 ret3 ret4 ret5 ret6 ret7 TRAP))

changeCWP1 <- And (Not WAIT) (op1=call)

changeCWP2t <- And (Not WAIT) (op1=ret1 ret3 ret5 ret7)

TRAP* <- Not (op1=TRAP)
Xcplal

RD_WR1* <- Not (op1=store0 store1 store2 store3 store4 store5 store6 store7)

storeSXT <- op1=store storem

pbusSHADOW* <- op1=load0 load1 load2 load3 load4 load5 load6 load7 store0
store1 store2 store3 store4 store5 store6 store7

pbusLtoINB* <- Not (Or (Not Alzeroforce*) (Not pSXTtobusL*))

pSXTtobusL* <- Not (And CPIPE1s<8> (Not (op1=call jmp store0 store1
store2 store3 store4 store5 store6 store7)))

pLOADLtobusL* <- Not (op1=store0 store1 store2 store3 store4 store5 store6
store7)

pSTOREwrite* <- Not (op1=store0)

byteEX <- op1=extract

EX_INSpass <- Not (op1=extract insert)

selBIbarl <- op1=sub storem loadm skip trap1 trap2 trap3 trap4 trap5 trap6
trap7 load1 load2 load3 load4 load5 load6 load7 store1 store2 store3 store4 store5
store6 store7
selaluOR* <- Not (op1=or)

selaluSR* <- Not (op1=sra srl)

selaluAND* <- Not (op1=and)

selaluXOR* <- Not (op1=xor)

selaluSUM* <- op1=extract insert xor or and sra srl

CPIPE1step* <- Not (Or (Not RESET*) (Not (op1=load1 load2 load3 load4 load5 load6 load7 store1 store2 store3 store4 store5 store6 store7 ret0 ret1 ret2 ret3 ret4 ret5 ret6 ret7 load loadc loadm store storem TRAP)))

CPIPE1loadc* <- Not (And RESET* (op1=load loadc))

CPIPE1store* <- Not (And RESET* (op1=store))

CPIPE1loadm* <- Not (And RESET* (op1=loadm load1 load2 load3 load4 load5 load6 load7))

CPIPE1storem* <- Not (And RESET* (op1=storem store1 store2 store3 store4 store5 store6 store7))

predecodeEA <- op1=load loadc store
aluCINbar <- Not (op1=sub loadm storem load1 load2 load3 load4 load5 load6 load7 store1 store2 store3 store4 store5 store6 store7 skip trap1 trap2 trap3 trap4 trap5 trap6 trap7 call jmp)

LD <- op1=load0 load1 load2 load3 load4 load5 load6 load7 store0 store1 store2 store3 store4 store5 store6 store7
\[
\text{pLOADwrite} \leftarrow \text{op2=} \text{load0 load1 load2 load3 load4 load5 load6 load7}
\]

\[
\text{nillonreturn} \leftarrow \text{op2=} \text{ret2 ret3 ret6 ret7}
\]

\[
\text{writeRFaccess2} \leftarrow \text{Not (op2=} \text{store storem store0 store1 store2 store3 store4 store5 store6 store7 load loadc loadm ret0 ret1 ret4 ret5 flush SKIP skip trap1 trap2 trap3 trap4 trap5 trap6 trap7 jmp)}
\]

\[
\text{busDtobusA2*} \leftarrow \text{Not (op2=} \text{call TRAP add sub and or xor sll srl sra insert extract ret2 ret3 ret6 ret7)}
\]

\[
\text{pbusDtoINA*} \leftarrow \text{Not (op2=} \text{load loadc loadm store storem load1 load2 load3 load4 load5 load6 load7 store1 store2 store3 store4 store5 store6 store7)}
\]

\[
\text{DSTtobusD2*} \leftarrow \text{Not (op2=} \text{add sub ret2 ret3 ret6 ret7 srl sra and or xor extract insert)}
\]

\[
\text{DSTvalid} \leftarrow \text{Not (op2=} \text{skip trap1 trap2 trap3 trap4 trap5 trap6 trap7 flush store storem store7 store6 store5 store4 store3 store2 store1 store0 TRAP SKIP call jmp)}
\]

\[
\text{opc2load} \leftarrow \text{op2=} \text{load0}
\]

\[
\text{lastPCtobusD1*} \leftarrow \text{Not (op2=} \text{TRAP call)}
\]
Alzeroforce* <- Not (op1=jmp call)

readRFaccessA1* <- Not (And Alzeroforce* (Not (And SRCvalid DSTvalid SRC1equalDST2 (Not SRC1equal16))))

readRFaccessB1* <- Not (And Alzeroforce* (Not (And SRCvalid DSTvalid SRC2equalDST2 (Not SRC2equal16))))

Alzero1* <- Not (Or (Not Alzeroforce*) (And SRCvalid SRC1equal16))

busDtobusAa* <- Not (Or (Not preadSWPtoA*) (Not preadTBtoA*) (Not preadPCtoA*) (And SRCvalid DSTvalid (Not opc2load) SRC1equalDST2 (Not SRC1equal16)))

pForwardtoINB* <- Not (And SRCvalid DSTvalid (Not opc2load) SRC2equalDST2 (Not SRC2equal16))

DSTtobusDa2* <- Not (Or (Not pbusDtoINA*) (Not pForwardtoINB*) (And SRCvalid DSTvalid (Not opc2load) SRC1equalDST2 (Not SRC1equal16)))

preadPCtoA* <- Not (And SRCvalid SRC1m1<4> (Not SRC1m1<3>) (Not SRC1m1<2>) (Not SRC1m1<1>) SRC1m1<0>)

preadTBtoA* <- Not (And SRCvalid SRC1m1<4> (Not SRC1m1<3>) SRC1m1<2> (Not SRC1m1<1>) SRC1m1<0>)
preadSWPtoA* <- Not (And SRCvalid SRC1m1<4> (Not SRC1m1<3>))
SRC1m1<2> (Not SRC1m1<1>) (Not SRC1m1<0>))

SRC1equal16 <- And SRC1m1<4> (Not SRC1m1<3>) (Not SRC1m1<2>)
(Not SRC1m1<1>) (Not SRC1m1<0>)

SRCvalid <- Not (Or (op1=jmp call) (Not pbusDtoINA*))
LoadforwtoINA* <- Not (And (Not SRC1equal16) SRC1equalDST2 opc2load DSTvalid SRCvalid)

LoadforwtoINB* <- Not (And (Not SRC2equal16) SRC2equalDST2 opc2load DSTvalid SRCvalid)

pSHAtobusA* <- Not (And SRCvalid SRC1m1<4> (Not SRC1m1<3>) (Not SRC1m1<2>) SRC1m1<1> SRC1m1<0>)

pSHBtobusA* <- Not (And SRCvalid SRC1m1<4> (Not SRC1m1<3>) (Not SRC1m1<2>) SRC1m1<1> (Not SRC1m1<0>))

pbusStobusA* <- Not (Or (Not preadPSWtoA*) (Not preadCWPtoA*))

preadPSWtoA* <- Not (And SRCvalid SRC1m1<4> (Not SRC1m1<3>) SRC1m1<2> SRC1m1<1> SRC1m1<0>)

preadCWPtoA* <- Not (And SRCvalid SRC1m1<4> (Not SRC1m1<3>) SRC1m1<2> SRC1m1<1> (Not SRC1m1<0>))
Apla2

\[ \text{writetoSHA1}^* \leftarrow \text{And (DSTvalid DST2}<4\rangle (\text{Not DST2}<3\rangle) (\text{Not DST2}<2\rangle) DST2<1> DST2<0> \]  

\[ \text{writetoSHB1}^* \leftarrow \text{And (DSTvalid DST2}<4\rangle (\text{Not DST2}<3\rangle) (\text{Not DST2}<2\rangle) DST2<1> (\text{Not DST2}<0>)) \]  

\[ \text{p writetoPC}^* \leftarrow \text{Not (And DSTvalid DST2}<4\rangle (\text{Not DST2}<3\rangle) (\text{Not DST2}<2\rangle) (\text{Not DST2}<1\rangle) DST<0> \]  

\[ \text{p writetoTB}^* \leftarrow \text{Not (And DSTvalid DST2}<4\rangle (\text{Not DST2}<3\rangle) DST2<2> (\text{Not DST2}<1\rangle) DST2<0> \]  

\[ \text{p writetoSWP}^* \leftarrow \text{Not (And DSTvalid DST2}<4\rangle (\text{Not DST2}<3\rangle) DST2<2> (\text{Not DST2}<1\rangle) (\text{Not DST2}<0>)) \]  

\[ \text{writetoPSW1}^* \leftarrow \text{Not (And DSTvalid DST2}<4\rangle (\text{Not DST2}<3\rangle) DST2<2> DST2<1> DST2<0> \]  

\[ \text{writetoCWP1}^* \leftarrow \text{Not (And DSTvalid DST2}<4\rangle (\text{Not DST2}<3\rangle) DST2<2> DST2<1> (\text{Not DST2}<0>)) \]
TAGtrap <- And tCPIPEls<6> (Or notanINT loadTRAP RXint)

notanINT <- Or (And (Not tCPIPEls<8>) (top1=sll srl sra add sub xor and or skip trap1 trap2 trap3 trap4 trap5 trap6 trap7) (Or tbusA<31> (Not tbusB<31>))) (And tCPIPEls<8> (top1=sll srl sra add sub xor and or skip trap1 trap2 trap3 trap4 trap5 trap6 trap7) tbusA<31>))

loadtrap <- And (top1=load loadc) (Or (And (Not tbusA<31>) (Or (And tbusB<31> (Not tCPIPEls<8>) tCPIPEls<8>) (And (Not tCPIPEls<8>) (Not tbusB<31>) tbusA<31>)))

RXint <- And (Not tbusA<31>) (top1=store)

pov_unflow <- And tCPIPEls<6> (top1=sub add sll)

trapinstr* <- Not (top1=trap1 trap2 trap3 trap4 trap5 trap6 trap7)

GStrap <- And tCPIPEls<6> (Or S1older nonLIFO RDcontext)

S1older <- And (Not tagcompare) (top1=store)

nonLIFO <- And tbusA<31> (top1=ret0 ret1 ret2 ret3 ret4 ret5 ret6 ret7)

RDcontext <- And (top1=store) (Not tbusB<28>) (Not tbusB<29>) (Not tbusB<30>) (Not tbusB<31>)
skipCONDenable* <- top1=skip
shiftAbus30 <- Or (And CPIPE1s<6> (op1=sra) Ain<30>) (And (Not CPIPE1s<6>) (op1=sra srl) Ain<31>)

shiftAbus31 <- Or (And CPIPE1s<6> (op1=sra srl) Ain<31>) (And (Not CPIPE1s<6>) (op1=sra) Ain<31>)
Tpla2

\( \text{TRAPreason}^0 \leftarrow \text{And} \ (\text{Not illegalopc}) \ (\text{Or} \ (\text{And} \ \text{IOINT} \ (\text{Not instrpagefINT}) \\
(\text{Not validtrap}) \ (\text{Not winunderflow}) \ (\text{Not SWI}) \ (\text{And} \ \text{GStrap} \ (\text{Not validtrap}) \\
(\text{Not winunderflow}) \ (\text{Not SWI})) \ (\text{And} \ \text{datapagefINT} \ (\text{Not winunderflow}) \ (\text{Not SWI})) \ (\text{And} \ \text{winoverflow} \ (\text{Not SWI})) \ \text{intTAGtrap}^* \)

\( \text{TRAPreason}^1 \leftarrow \text{And} \ (\text{Not} \ (\text{Or} \ \text{illegalopc} \ (\text{Not intTAGtrap}^*))) \ (\text{Or} \ (\text{And} \ \text{GStrap} \ (\text{Not datapagefINT}) \ (\text{Not winunderflow})) \ (\text{And} \ \text{validtrap} \ (\text{Not datapagefINT}) \ (\text{Not winunderflow})) \ \text{winoverflow} \ \text{SWI})

\( \text{TRAPreason}^2 \leftarrow \text{And} \ (\text{Not} \ (\text{Or} \ \text{winoverflow} \ \text{SWI} \ (\text{Not intTAGtrap}^*) \ \text{illegalopc})) \ (\text{Or} \ \text{GStrap} \ \text{validtrap} \ \text{datapagefINT} \ \text{winunderflow})

\( \text{TRAPreason}^3 \leftarrow \text{And} \ (\text{Not} \ (\text{Or} \ \text{GStrap} \ \text{validtrap} \ \text{datapagefINT} \ \text{winunderflow} \ \text{winoverflow} \ \text{SWI} \ (\text{Not intTAGtrap}) \ \text{illegalopc})) \ (\text{Or} \ \text{instrpagefINT} \ \text{IOINT}) \)
ptagcompare <- Or busB<31> (Not busA<31>) busA<30> · (And busB<30> (Not busA<30>) (Or (And (Not busB<29>) (Not busA<29>)) (And (Not busB<28>) (Not busA<28>) (Not busB<29>)) (And (Not busA<28>) (Not busA<29>) (Not busB<28>)) (And (Not busA<29>) (Not busB<29>)))
Illpla

\[ \text{illegalopc <- Not (op1=flush SKIP TRAP ret0 ret1 ret2 ret3 ret4 ret5 ret6 ret7} \]
\[ \text{load0 load1 load2 load3 load4 load5 load6 load7 store0 store1 store2 store3 store4} \]
\[ \text{store5 store6 store7 srl sra insert extract add sll sub or xor and skip trap1 trap2} \]
\[ \text{trap3 trap4 trap5 trap6 trap7 load loadm loadc store storem call jmp)} \]

Condpla

aluVout <- And cselaluSUM (Or (And (Not cAinMSB) (Not cBinMSB) caluMSB) (And cAinMSB cBinMSB (Not caluMSB)))

aluCout <- And cselaluSUM (Or (And cAinMSB cBinMSB) (And cAinMSB (Not caluMSB)) (And cBinMSB (Not caluMSB)))

aluSout <- caluMSB

pCONDvalid* <- Not (Or (And caluZout (d1=04)) (And (Not caluZout) (d1=05)) (And (Or (And aluSout (Not aluVout)) (And (Not aluSout)) aluVout) (d1=02)) (And (Not (Or (And aluSout (Not aluVout)) (And (Not aluSout) aluVout))) (d1=03)) (And (Or (And aluSout (Not aluVout))) (And (Not aluCout)) (dl=12)) (And aluCout (dl=13)) (And (Or (Not aluCout) caluZout) (d1=16)) (And (Not (Or (Not aluCout) caluZout)) (d1=17)) (d1=01) (And (Or (Not aluCout) caluZout) (Not cAin0) (d1=22)) (And (Or (Not (Or (Not aluCout) caluZout)) cAin0) (d1=23)))
Appendix D
Input/Output Timing Specifications

Loading for these measurements was the chip package and a 10pF scope probe.

Diagram:
- PHI1*
- PHI2*
- PHI3*
- FSHCNTL
- I/D*
- RD/WR*
- A00-A27

- D00-D31 (out)
- D00-D31 (in)
- RESETin*
- PAGEin*
- IOin*
- WAIT*
WAIT* sample
valid due to the shown sample