Formal Verification of Processors and Low Level Software

W. J. Paul

wjp@cs.uni-sb.de

May 25, 2002

Joint work with

Berg, Beyer, Even, Hillebrand, Jacobi, Knuth,

Kröning, Leister, Müller, Preiß, Seidel

MOTIVATION FOR FORMAL VERIFICATION

- lives (or worse money) depend on correctness
- classical debugging of systems too slow (2^{128} different inputs...)
- classical debugging of proofs too slow (decades)
- formal verification is not terribly slow

A WORD ON TOOLS FOR VERIFICATION

- verification systems capable of verifying real computer systems exist
  - not easy to use
  - high salary ...
- verification sometimes slow because high school mathematics etc. not yet formalized (temporary)
- model checking (checking 2^{32} but not 2^{64} cases) fast special case of theorem proving

THE MAIN TECHNICAL PROBLEMS IN VERIFICATION OF REAL SYSTEMS

- reengineer computer systems in a structured way
  (without it you still can model check ...)
- formulate correctness of modules
- proofs of lemmas usually easy (as usual)
- automate as much as possible fot real (not toy) systems

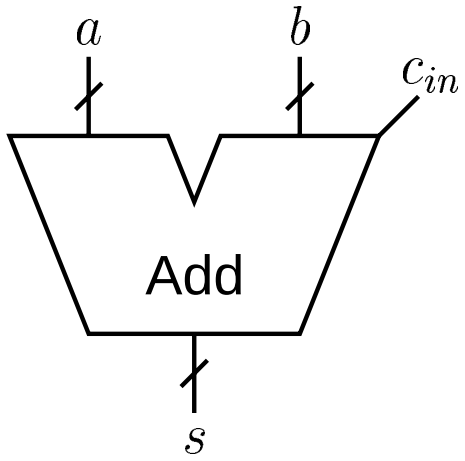
OVERVIEW

structuring the construction of a parallel processor

1. arithmetic circuits
2. sequential control
3. pipelining
4. forwarding and stalling
5. speculation
6. interrupts
7. out of order execution
8. caches
9. memory management units
10. virtual shared memory

1. ARITHMETIC CIRCUITS (INFORMATIK 2)

- numbers and their representation
- semantics of circuits (every node has a depth ...)



adder: $\langle a \rangle + \langle b \rangle + c_{in} = \langle s \rangle$

1. Arithmetic circuits (Computer Architecture)

- Def: finite set R of IEEE-representable numbers
- Def: $r : \mathbb{R} \to R \cup \{-\infty, \infty\}$ (Rounding)
- Spec: $\circ_{IEEE}(x, y) = r(x \circ y), \circ \in \{+, -, *, \ldots\}$
- exceptions ...
- 220 pages of NICE mathematics

2. SEQUENTIAL CONTROL (INFORMATIK 2)

- construction: finite precomputed control (tens of simple boolean circuits)
- correctness:
  - implementation machine: the hardware
  - specification machine: from specification of machine language

example: add

$RD := RS1 + RS2; PC = PC + 4$ shorthand for

$GPR(RD) := GPR(RS1) + GPR(RS2)$ shorthand for

$[GPR(RD)] := [GPR(RS1)] + [GPR(RS2)] \pmod{32}$ shorthand for

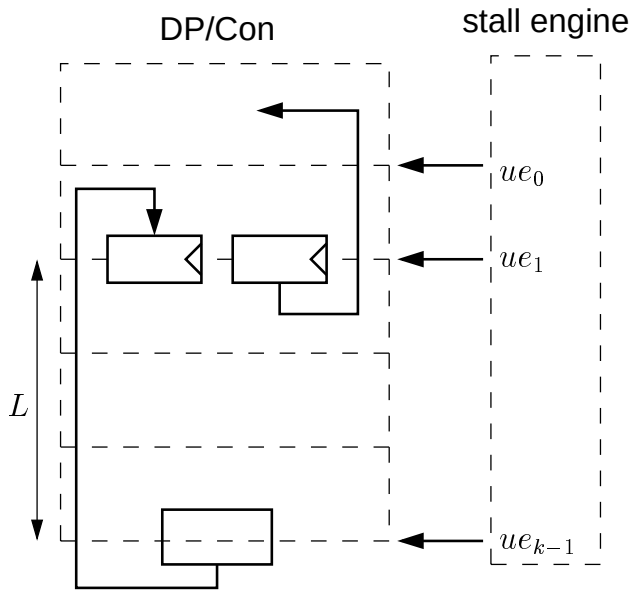
$[c^{i+1}.GPR(RD)] = [c^i.GPR(RS1)] + \ldots \pmod{32}$,

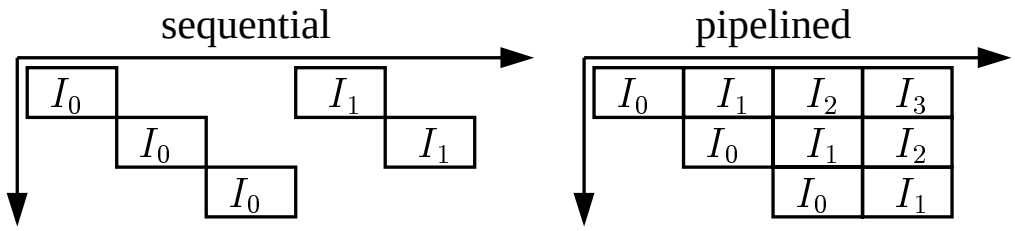
$RS1 := \langle M(c^i.PC)[25:21] \rangle$

- Simulation theorem: computer algebra

3. PIPELINING

- DP/Con: k-stage data path and precomputed control
- L: length of longest backward edge
- ue_k: update enable signal for stage k





3. PIPELINING

Correctness:

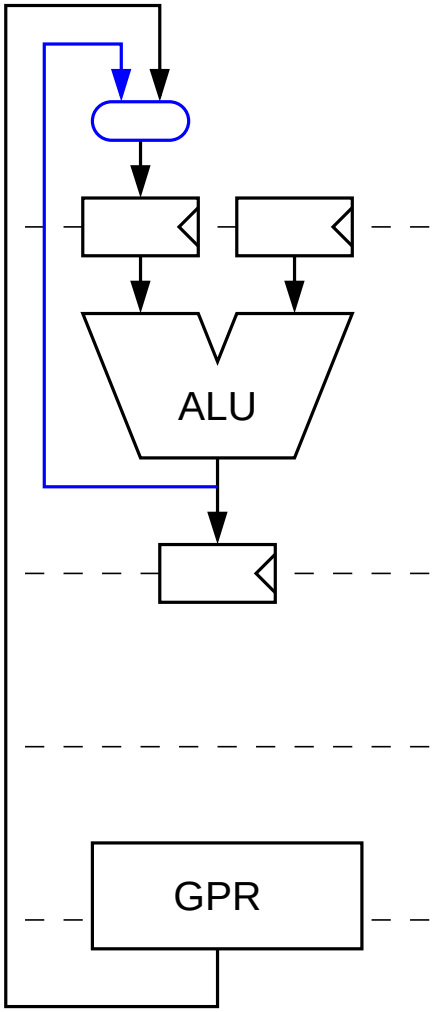
Hypothesis: I_i generates data $\Rightarrow$ $I_{i+1}, \ldots I_{i+L}$ do not use it.

Statement: M_{pipe} simulates M_{seq}

Proof:

- $\forall T\ \forall k$: pipelined stage k has in cycle T same input as seq stage k in corresponding (scheduling) cycle.
- Induction on T.
- corresponding stages have in corresponding cycles (scheduling) identical inputs

4. Forwarding and Stalling

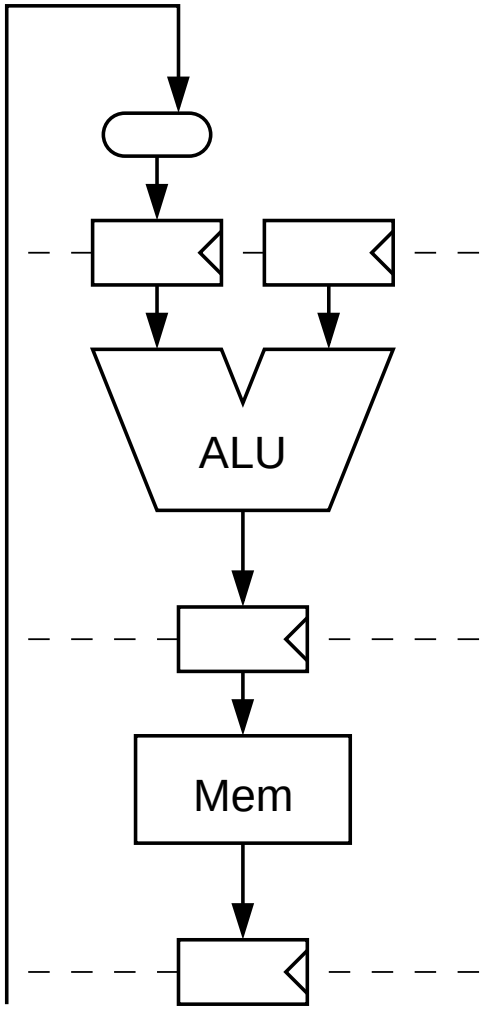


R4 = R1 + R5

R1 = R2 + R3

Forwarding

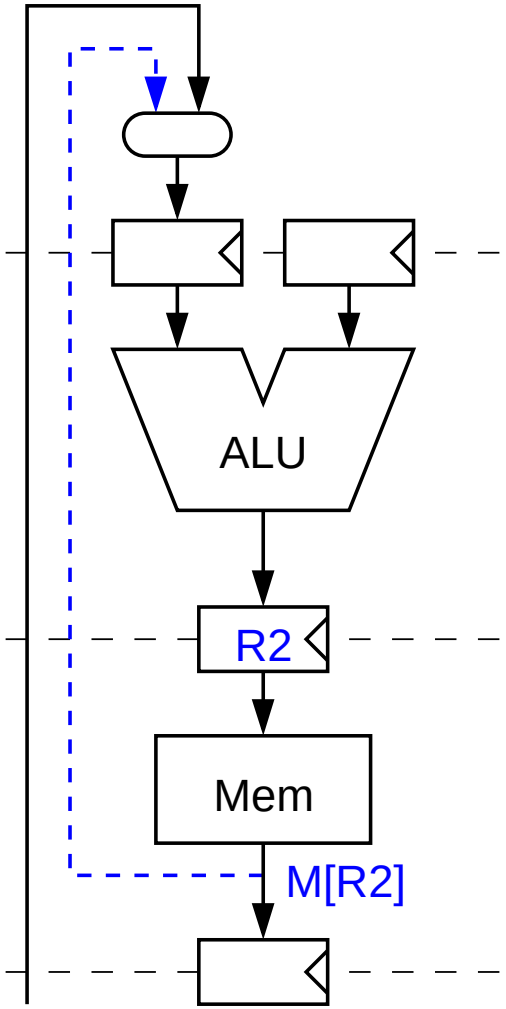
4. Forwarding and Stalling



R4 = R1 + R5

R1 = M[R2]

4. Forwarding and Stalling



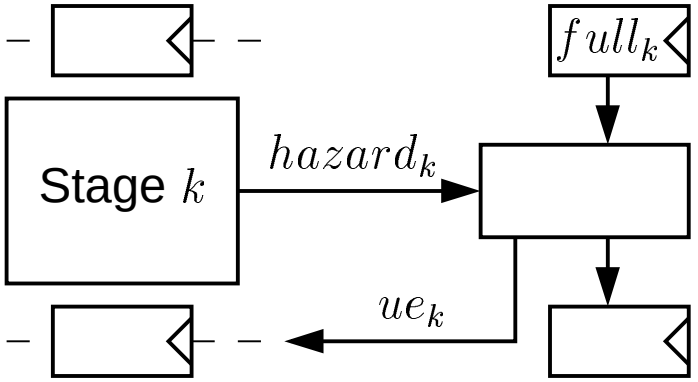
stalling

R4 = R1 + R5

empty

R1 = M[R2]

4. Forwarding and Stalling



- $hazard_i$: stage i should not be updated
- $full_i$: stage i has data

$$
\begin{aligned}
stall_i &:= full_i \wedge (hazard_i \vee stall_{i+1}) \\
ue_i &:= full_i \wedge \overline{stall_i} \\
full_i &:= ue_i \vee (full_i \wedge \overline{ue_{i+1}})
\end{aligned}
$$

(2000)

4. Forwarding and Stalling

Correctness:

Statement: M_{pipe} simulates M_{seq}

Proof: $\forall T\, \forall k$: pipelined stage k has in cycle T same input as sequential stage k in corresponding (scheduling) cycle.
$T \rightarrow T + 1$: in order K, $K - 1$, $K - 2$, . . .

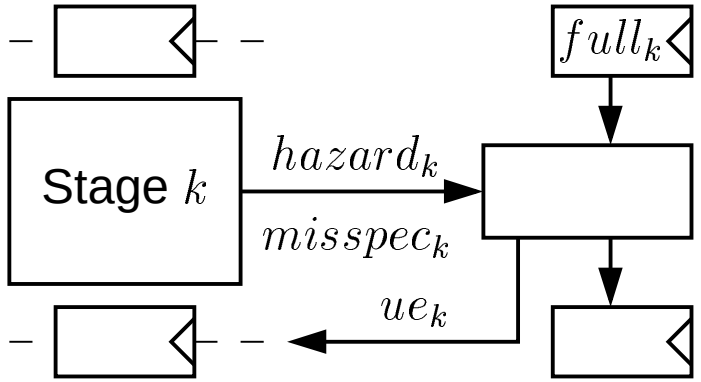
Remark: superpipelining:

- pipeline RAMs
- reduce circuit depth of stall engines to O(1) indep. of K
- pipeline forwarding

5. SPECULATION

guess future data, say in stage 1

- predict branch not taken
- predict there will be no interrupt



$misspec_i$: misspeculation detected in stage i

$$
\begin{aligned}
flush_i &:= misspec_i \vee flush_{i+1} \\
full_i &:= \text{— old —} \wedge \overline{flush_i} \\
restart &:= flush_1
\end{aligned}
$$

Correctness: 2 cases in simulation theorem (flush/no flush)

6. (Nested maskable precise) Interrupts

very complex

restart:

- save (old or new) interrupt mask bits
- save interrupt cause
- save exception data
- mask all interrupts
- save old PC (or short history of PC's)
- PC := SISR (start interrupt service routine)

similar rfe (return from exception) instruction

- restore interrupt mask bits.

6. (NESTED MASKABLE PRECISE) INTERRUPTS

complication:

- special purpose register file (for save, restore)
- move instruction between general purposa and special registers
- nontrivial theory of forwarding (1998-2000)

Correctness of hardware: no further complication

6. (NESTED MASKABLE PRECISE) INTERRUPTS

Correctness of specification:

Hypothesis: 2 pages specification for interrupt service routines

Statement: jump to interrupt service routine is 'like' procedure call

Formalization: semantics of programming languages

Proof: show that calls and returns are atomic (except reset)
9 pages mathematics

formal proof ? today no
would you bet your head on it ??

7. Out of order execution

Tomasulo scheduler

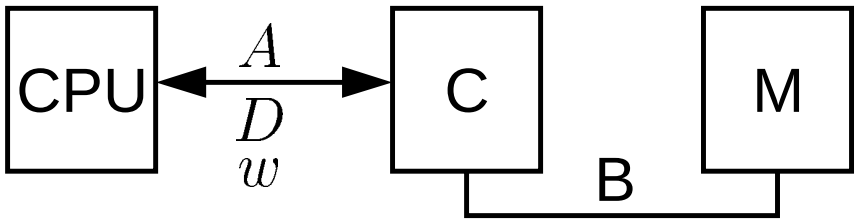
- classical benchmark problem for verification
- usually simplified
- usually not synthesizable hardware
- usually not integrated into entire processor
- liveness nontrivial

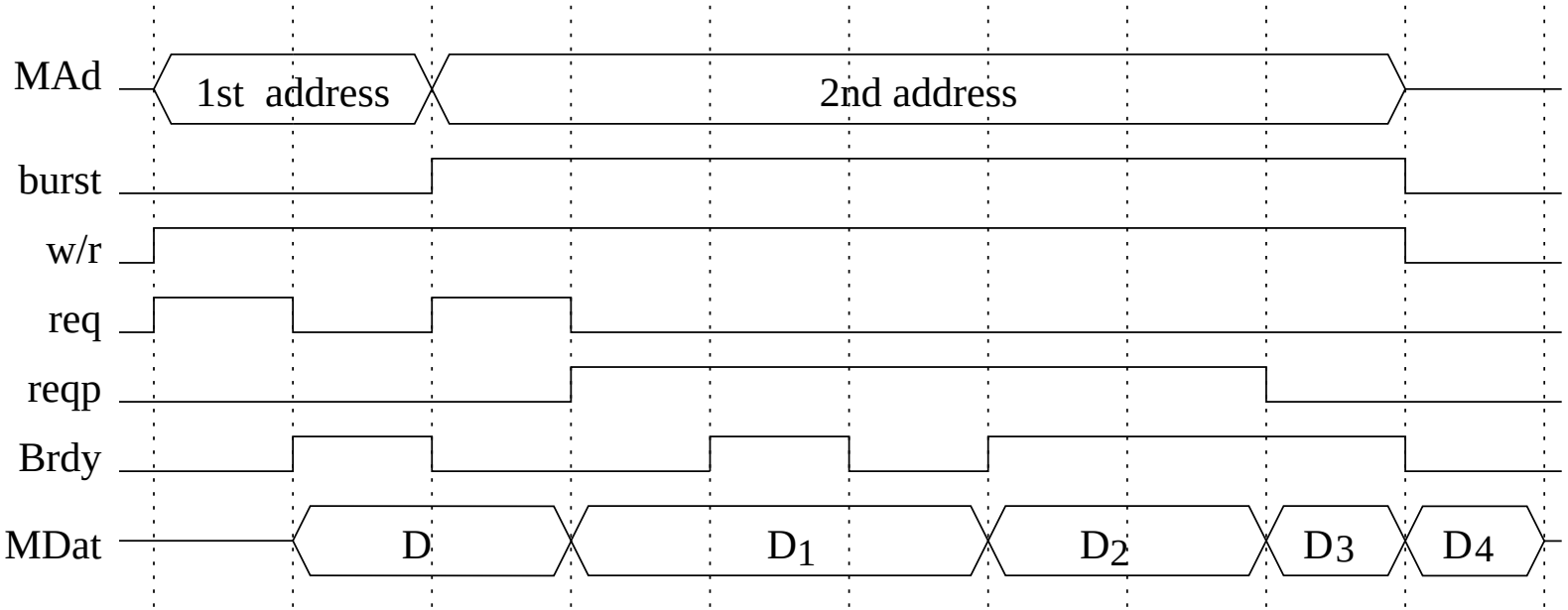
8. CACHES

M: main memory, big and slow

C: cache, small and fast

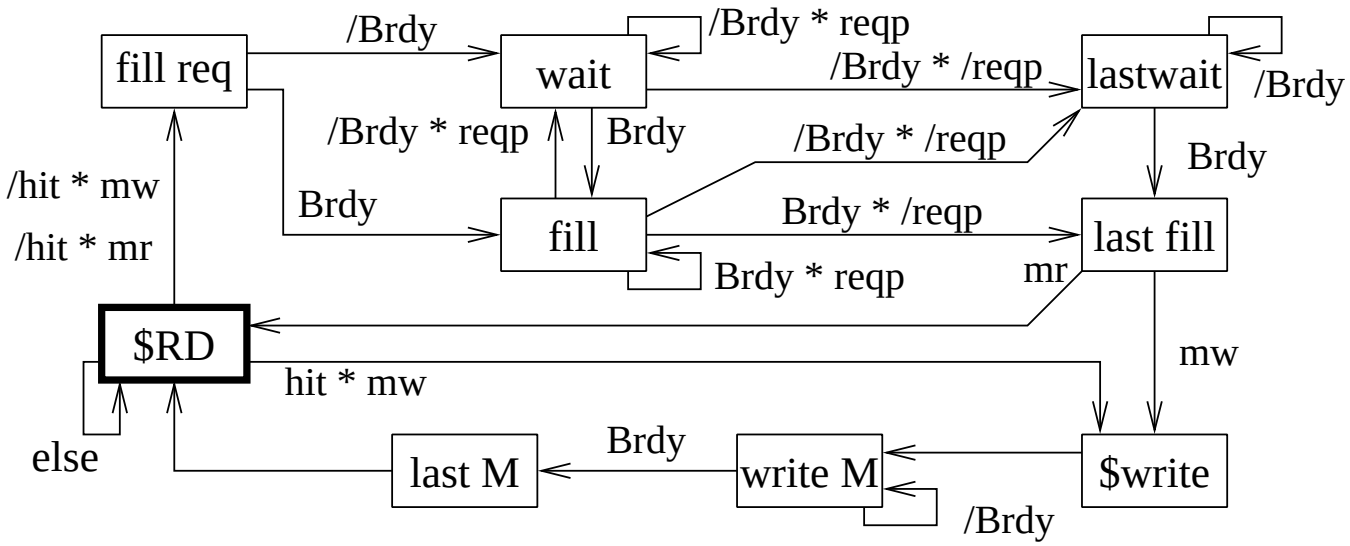
B: bus!!, has protocol





8. CACHES

- Automata for cache, main memory implement protocol

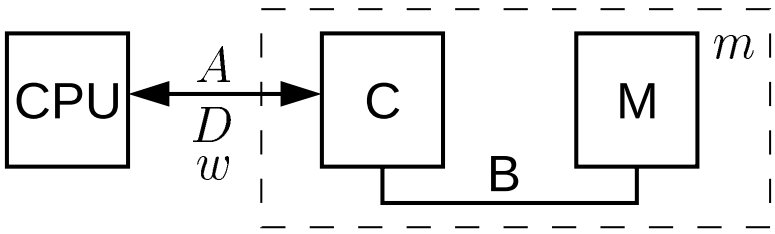


Lemma (complex!): automata implement protocol

Proof: presently theorem proving

better model checking ?

8. CACHES



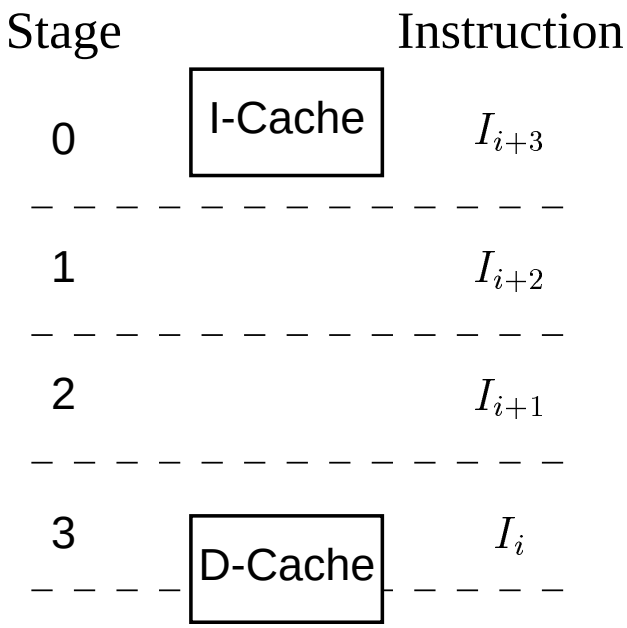
Correctness statement (extremely simple!):

Cache + main memory behave like uniform memory

$$m(x) := \begin{cases} D & \text{if } x = \langle A \rangle \text{ and } w = 1 \\ m(x) & \text{otherwise} \end{cases}$$

8. CACHES

- instruction cache (stage 0)
- data cache (stage 3)



cache coherence requires speculation:
no self modifying code within 3 instructions

$$I_i, I_{i+1}, I_{i+2} \text{ do not write } m(c^{i+2}.PC)$$

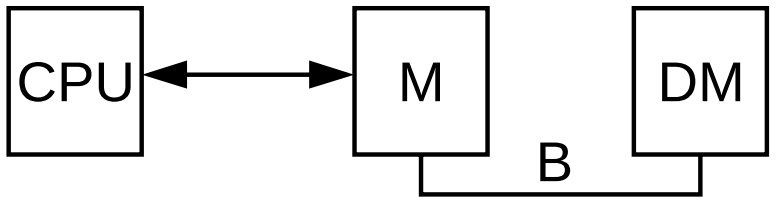
9. MEMORY MANAGEMENT UNITS

Tasks:

- isolation of user tasks
- memory relocation (zero copy)
- virtual memory

M: main memory, small, fast

DM: disk memory, large, slow



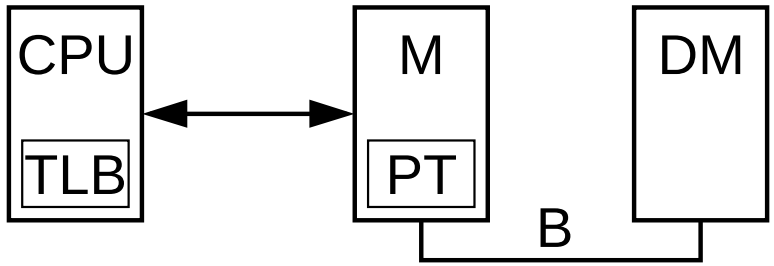
9. Memory management units

M: is cache for DM

PT: page table,

'cache directory' for M

TLB: table look aside buffer, cache for PT



'cache miss' in M $\leadsto$ page fault interrupt

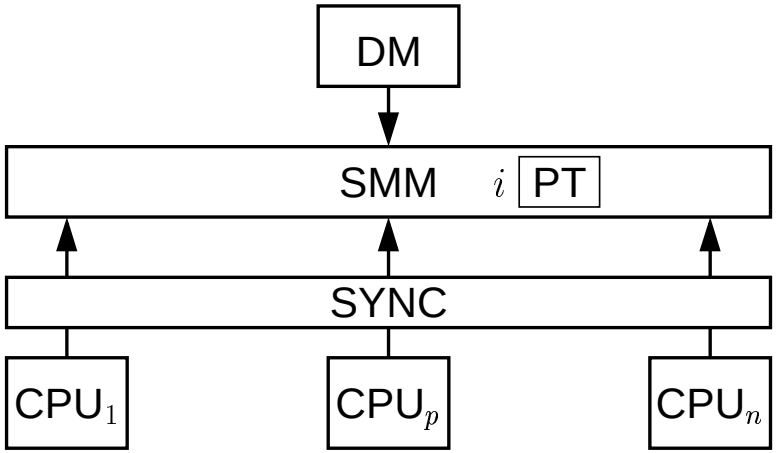
$\leadsto$ interrupt service routine (ISR) $\leadsto$ rfe

Correctness statments:

- TLB, PT: CPU sees uniform page table
- M, VM, ISR: user task sees uniform virtual memory

Complicated correctness of interrupt mechanism is lemma, hidden (!!!) in correctness stament of M

10. VIRTUAL SHARED MEMORY



SMM: shared main memory, sequentially consistent

(cache based or SB-PRAM)

SYNC: tracks $CP = \{(p, i) |$ processor p currently acceses page $i\}$

ISR for page fault plans to evicts page i

$\rightsquigarrow$ SYNC stalls ISR until all $(p, i) \in CP$ have completed access

$\rightsquigarrow$ ISR invalidates PT(i) $\rightsquigarrow$ ...

Correctness: processors see sequentially consistent virtual shared memory

CONCLUSION

This opens the way of proving the correctness of a mainframe!