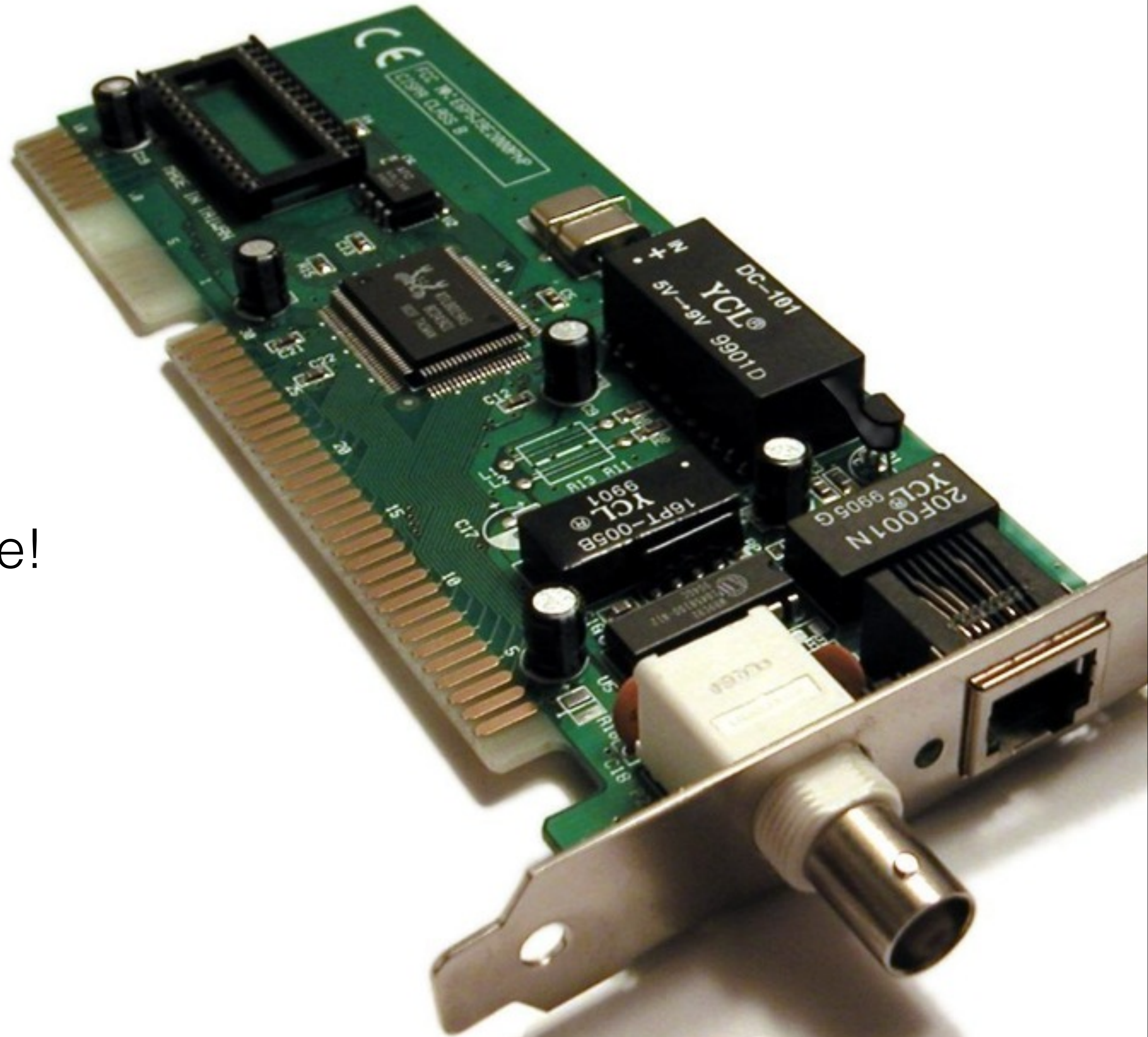
Quo Vadis, ISA & Cui Bono?

Michael Engel – TU Dortmund

GI FG-BS – TU Berlin – 8.11.2013

ISA?

Not that one!

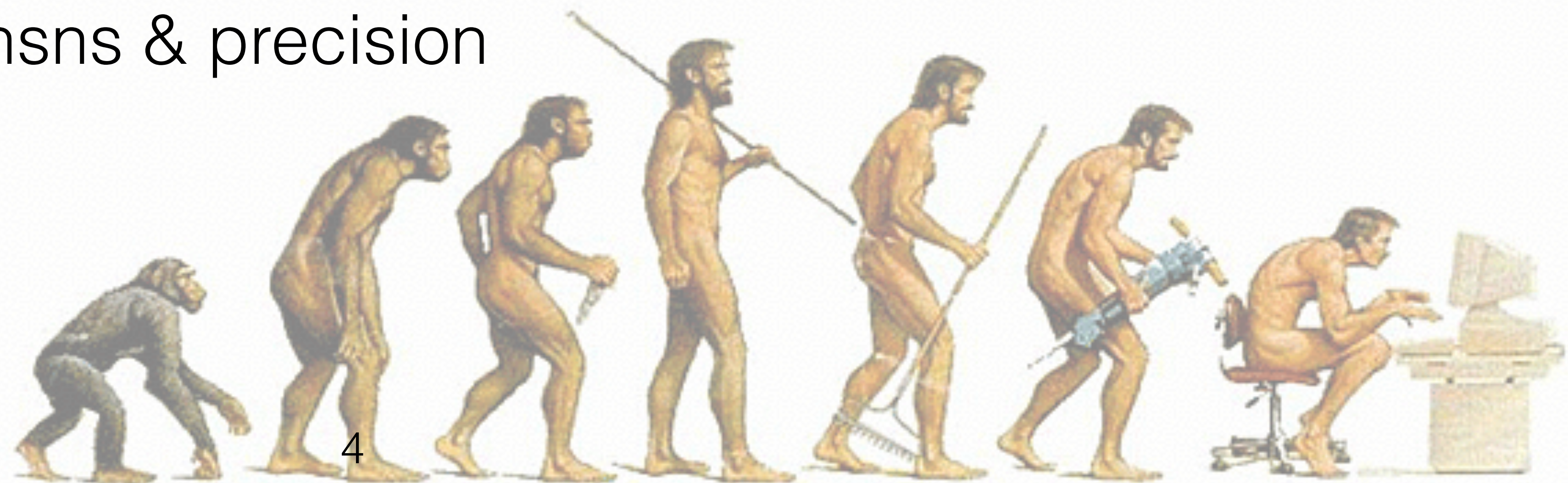


ISA!

- Instruction Set Architecture
- *"An [...] instruction set architecture (ISA) is the part of the computer architecture related to programming, including the native data types, instructions, registers, addressing modes, memory architecture, interrupt and exception handling, and external I/O.
An ISA includes a specification of the set of opcodes (machine language), and the native commands implemented by a particular processor."* [Wikipedia]
- Let's take a closer look on trends in ISA extensions...

Evolution of ISA Extensions

- Only considering (Intel) x86 architecture here
- 1978: Introduction of 8086 CPU architecture
- 1980: 8087 FPU
- 1982: 80286 – 16 bit protected mode
- 1985: 80386 – 32 bit protected mode
- 1996: MMX – SIMD
- 1999: SSE1, 2001: SSE2, 2004: SSE3, ...
- 2006: SSE4 – more insns & precision
- 2008: AES
- ...what else?



Quo Vadis, ISA?

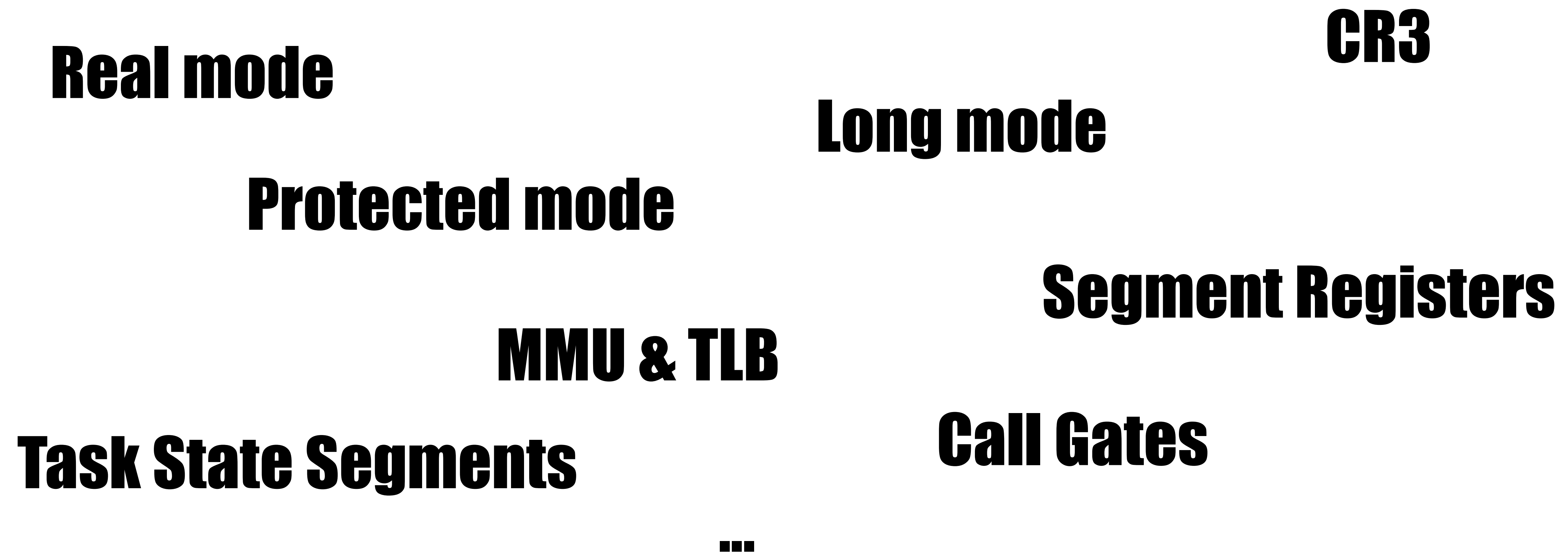
- Current developments in instruction set extensions
- A glimpse on future developments

… & Cui Bono?

- Whom are the ISA extensions expected to help?
- How can they help OS designers and developers?

- This talk: Mostly questions (few answers)
  - Starting point for discussions

ISA: No Fun for the OS?



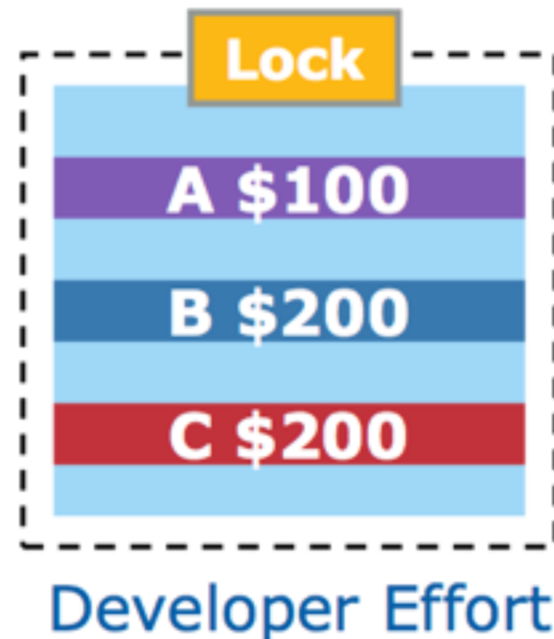
Processor designers are (often) giving OS designers and developers a hard time

Intel TSX

- Intel TSX: Transactional Synchronization Extensions
  - Implemented in Haswell and beyond
    - Beware: not in all Haswell CPUs (→ ark.intel.com)
- Transaction semantics for main memory accesses
- Implemented by buffering memory writes
- Hardware uses L1 cache to buffer transactional writes
  - Writes not visible to other threads until after commit
  - Eviction of transactionally written line causes abort
- Buffering at cache line granularity

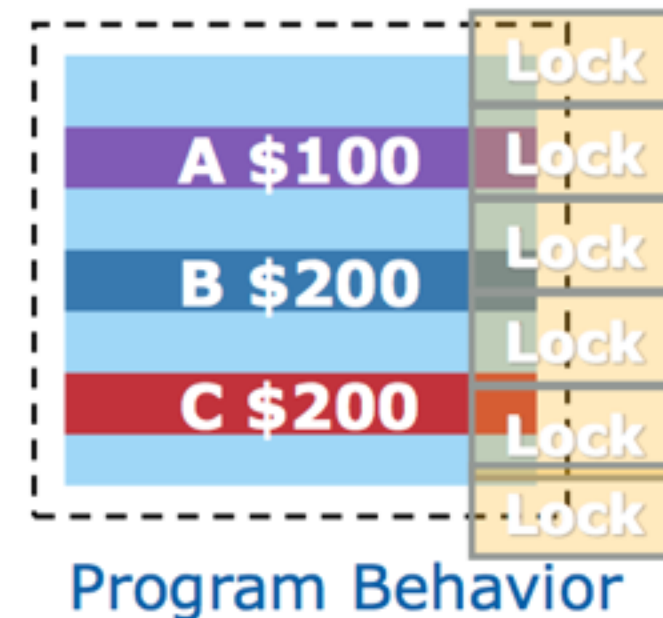
TSX Example: Lock Elision

Coarse grain locking effort



Hardware

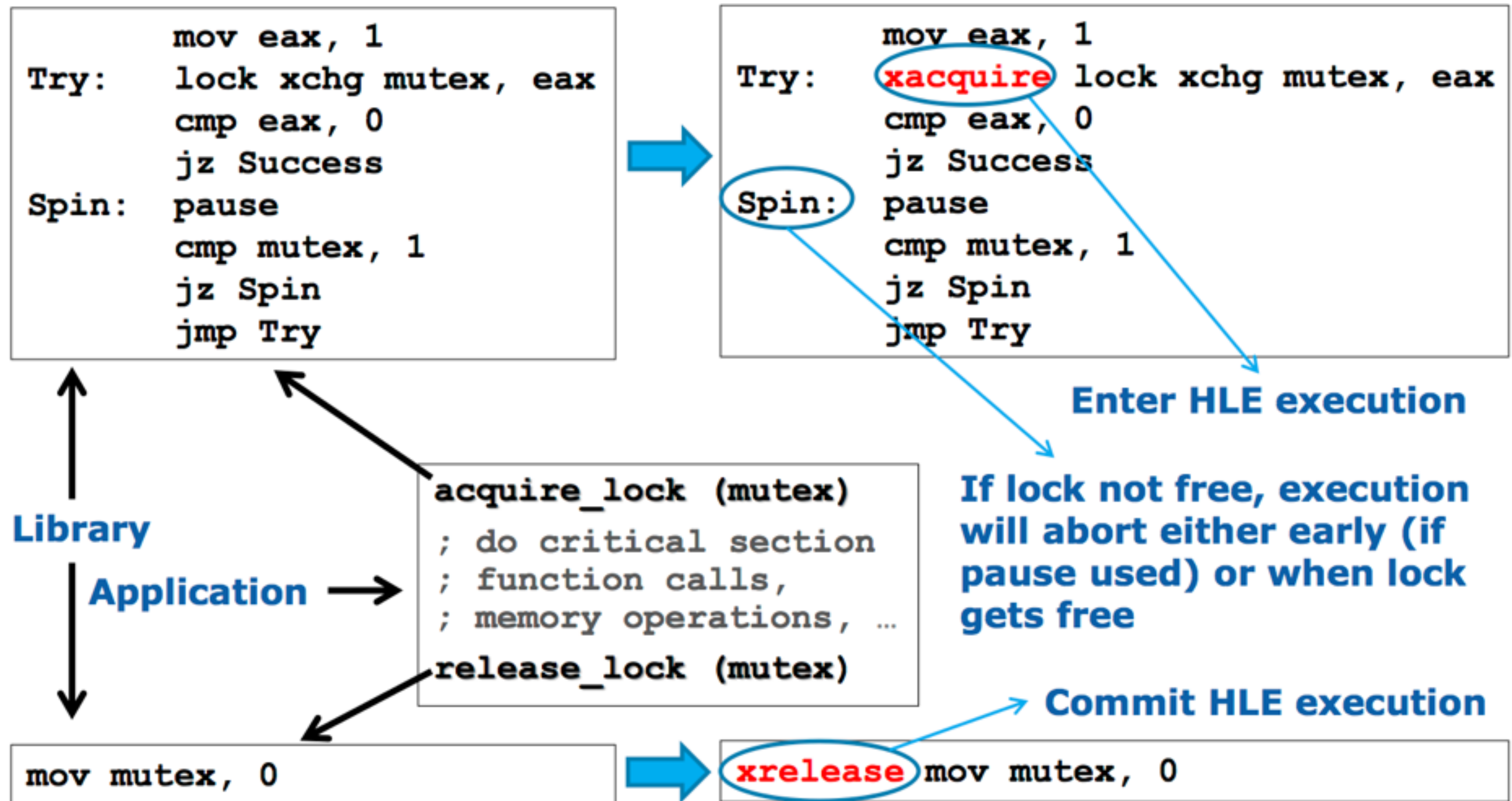
Fine grain locking behavior



- Developer uses coarse grain lock
  - Hardware elides the lock to expose concurrency
  - Alice and Bob don’t serialize on the lock
  - Hardware automatically detects real data conflicts

Ravi Rajwar, Martin Dixon (Intel): Intel Transactional Synchronization Extensions, IDF'12

TSX Example: Lock Elision



Ravi Rajwar, Martin Dixon (Intel): Intel Transactional Synchronization Extensions, IDF'12

TSX: RTM mode

```
Retry: xbegin Abort
       cmp mutex, 0
       jz Success
       xabort $0xff

Abort:
    … check EAX and do retry policy
    … actually acquire lock or wait
    … to retry.
    …
```

... Enter RTM execution, Abort is fallback path
... Check to see if mutex is free
... Abort transactional execution if mutex busy
... Fallback path in software
... Retry RTM or explicitly acquire mutex

```
acquire_lock (mutex)
; do critical section
; function calls,
; memory operations, …
release_lock (mutex)
```

RTM = "Restricted Transactional Memory"

```
cmp mutex, 0
jnz release_lock
xend
```

... Mutex not free →was not an RTM execution
... Commit RTM execution

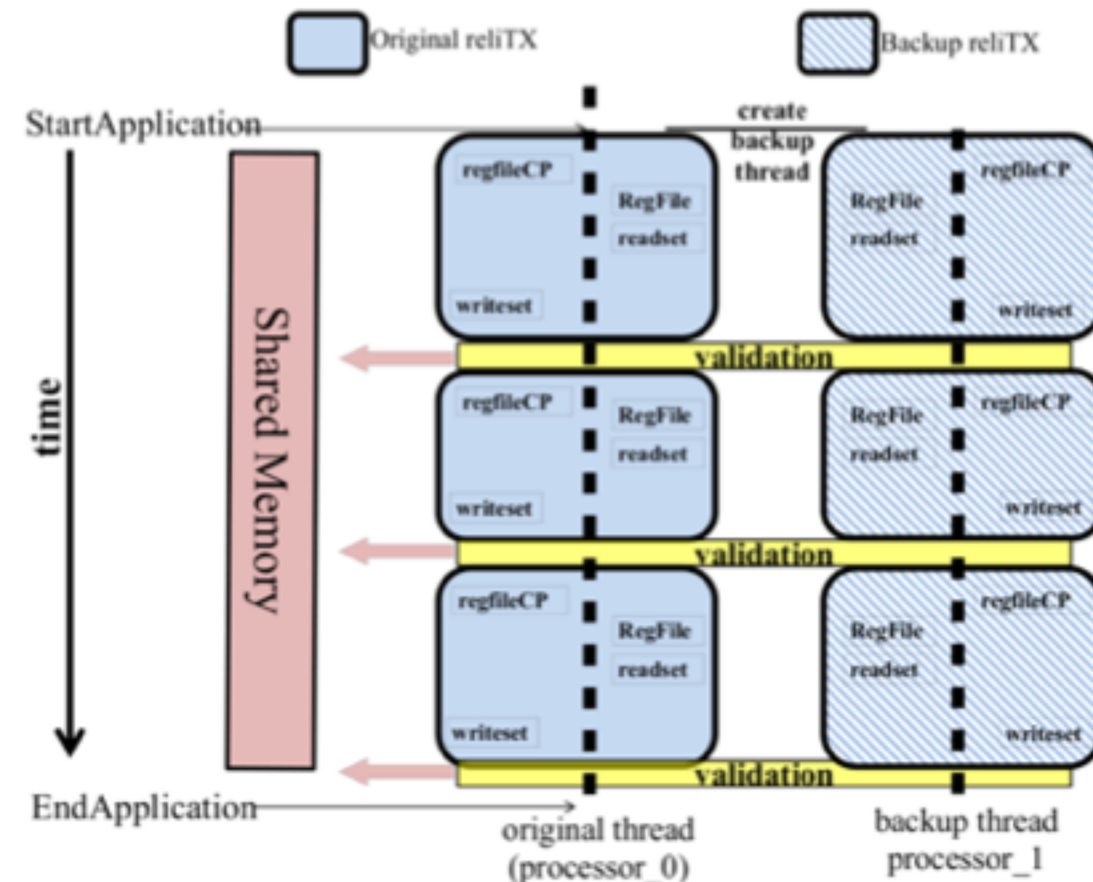
Ravi Rajwar, Martin Dixon (Intel): Intel Transactional Synchronization Extensions, IDF'12

Use Case: Checkpointing

- Dependability research – DFG SPP1500
- Checkpoint and recovery: common method to restore state corrupted by HW error
- Is TSX useful here?
  - Idea: Hardware TM enables "free" checkpointing and restore for fault-tolerant applications
  - Run thread+checker thread(s) in parallel on the same memory locations
    - If deviation detected, abort transaction and restore state
    - Otherwise, commit transaction and continue

...Research Ideas

- You have a great idea for a research topic...and what happens?
  - Someone else had that idea before!
    - Might have been obvious here?
- Yalcin [1] requires comparator HW
- Metzlaff [2] proposes comparison approach using lazy versioning
- What's left for you to do?
  - Evaluate if these ideas really work *on real hardware*



[1] Gulay Yalcin et al.: FaulTM: Error Detection and Recovery Using Hardware Transactional Memory, Proc. of DATE 2013, pp. 220–225

[2] Stefan Metzlaff, Sebastian Weis, and Theo Ungerer: Towards Transactional Memory for Safety-Critical Embedded Systems, Euro-TM WS on Transactional Memory 2013 (ext. Abstract)

...TSX Implementation

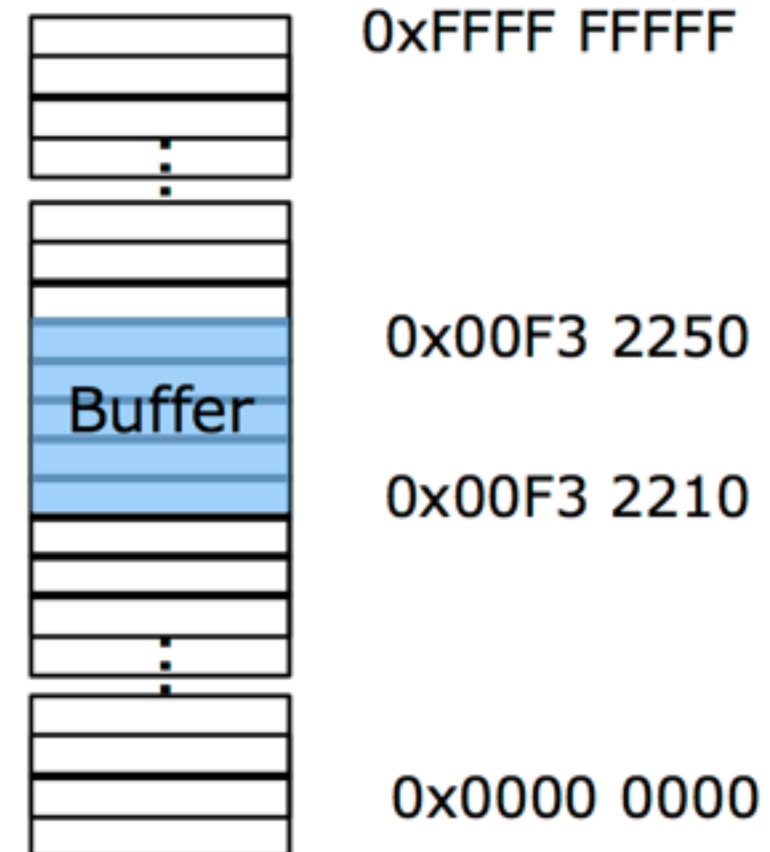
- Checkpoint/restore of (mostly) register and L1 data cache state
- Read and write addresses for conflict checking
  - Tracked at cache line granularity using physical address
- Data conflicts occur if at least one request is doing a write
  - Detected at cache line granularity
  - Detected using existing cache coherence protocol
  - Abort when conflicting access detected
- Restricted size of transactions
  - Depending on L1 D$ utilization, locking of cache lines, ...

...and Disillusions

- Problem with Intel TSX for dependability checkpointing support
  - Even if *identical data* is written by concurrent tasks (WAW conflict), the transaction is **aborted**!
- Additional complications:
  - Some instructions and events may cause aborts
    - Uncommon instructions, interrupts, faults, etc.
  - Software must provide a non-transactional path
    - HLE: Same software code path executed without elision
    - RTM: SW fallback handler must provide alternate path
- Best case: (lots) more work required
- Worst case: Intel TSX not useful for dependability checkpointing

Intel MPX

- New instructions enabling runtime buffer overflow checks
- Improve software security and robustness
- Four new registers to store bounds
- New instructions to check bounds prior to memory access
    - Exception on bound violations
- Expected 2015...



Upper Bound Lower Bound

e.g. BND0 = 00F3 2250 00F3 2210

Baiju Patel, Intel: Stop Buffer Overflows in Their Tracks with Intel Memory Protection Extensions (IDF'13 Presentation)

MPX strcpy

```
// s2 is RDX, and s1 in RCX, bounds for s1 in BND0 by calling
    convention
strcpy(char *s1, char *s2) {
  while (*s1++ = *s2++) {}
}

        BNDCL   BND0, [RCX] ; check s1 (RCX) LB against bounds in BND0
L1:     MOVB    RAX, [RDX] ; load a char
        INC     RDX
        BNDCU   BND0, [RCX] ; check UB for s1 before write
        MOVB    [RCX], AL
        INC     RCX
        TESTB   AL, AL
        BND JNE L1          ; BND (0xF2) prefix is NOP in MPX enabled code
        BND RET
```

New Register

MPX instructions

BNDCL and BNDCU check lower and upper bounds,
if check fails, signal exception #BR

Baiju Patel, Intel: Stop Buffer Overflows in Their Tracks with Intel Memory Protection Extensions.

Time to think about...

- Is there demand for OS-supporting ISA extensions?
- Can we improve the interaction between OS and processor architecture?
  - Perhaps: a fresh look at OS-CPU codesign?
- What might these extensions look like?
  - Inspiration from µcode? DEC Alpha PALcode

Don't ask what you can do
for the processor designer
–
ask what the processor
designer can do for you!

ISA and RISC-vs-CISC

- Patterson&Ditzel's paper: Foundation of RISC ideas
- Reduced instruction sets vs. "baroque" CISC ISA
- Classical argument in favor of RISC
  - VAX "Index" instruction: similar to proposed MPX

David Patterson, David Ditzel: The Case for the Reduced Instruction Set Computer
ACM SIGARCH Computer Architecture News, Vol. 8 Issue 6, Oct. 1980, pp. 25-33

VAX Index Instruction

- Similar to newly proposed x86 MPX extension

The **indexin** operand is added to the **subscript** operand and the sum multiplied by the **size** operand. The **indexout** operand is replaced by the result. If the **subscript** operand is less than the **low** operand or greater than the **high** operand, a subscript range trap is taken.

INDEX

Compute Index

Format

opcode subscript.rl, low.rl, high.rl, size.rl, indexin.rl, indexout.wl

Condition Codes

N	<—	indexout LSS 0;
Z	<—	indexout EQL 0;
V	<—	0;
C	<—	0;

Exceptions

subscript range

Opcodes

0A INDEX index

Compaq Computer Corporation: VAX MACRO and Instruction Set Reference Manual (2001) Order Number: AA-PS6GD-TE

What Patterson wrote

This case covers 40% of the load multiple instructions in typical programs. Another comes from the VAX-11/780. The INDEX instruction is used to calculate the address of an array element while at the same time checking to see that the index fits in the array bounds. This is clearly an important function to accurately detect errors in high-level languages statements. We found that for the VAX 11/780, replacing this single "high level" instruction by several simple instructions (COMPARE, JUMP LESS UNSIGNED, ADD, MULTIPLY) that we could perform the same function 45% faster! Furthermore, if the compiler took advantage of the case where the lower bound was zero, the simple instruction sequence was 60% faster. Clearly smaller code does not always imply faster code, nor do "higher-level" instructions imply faster code.

David Patterson, David Ditzel: The Case for the Reduced Instruction Set Computer
ACM SIGARCH Computer Architecture News, Vol. 8 Issue 6, Oct. 1980, pp. 25-33

...and how he was proven wrong

- Reaction of DEC's VAX architects
- One of the basic propositions for RISC was invalid

The paper reports that a sequence of several simple instructions can replace the VAX INDEX instruction with a 45% speed gain on the 780. This is a problem of implementation, not architecture. Fundamentally, after all, the implementation of the INDEX function with more than one instruction simply cannot take less time than the one-instruction version, assuming equal hardware in both cases. The explanation of this anomaly is that the 780's Floating Point Accelerator speeds up the multiply in the multi-instruction implementation, but doesn't see INDEX at all.

Douglas W. Clark and William D. Strecker: Comments on "the case for the reduced instruction set computer," by Patterson and Ditzel
ACM SIGARCH Computer Architecture News, Vol. 8 Issue 6, Oct. 1980, pp. 34-38

A Proof (No Pudding)

- You thought you would never see microcode again? :-)

```
ZZ-ESOAA-124.0  ; INDEX .MIC [600,1204]     Index instruction  14-Jan-82        Fiche 2  Frame L5        Sequence 269
; P1W124.MCR 600,1204]       MICRO2  1L(03)    14-Jan-82  15:30:16    VAX11/780 Microcode : PCS 01, FPLA 0E, WCS124
; INDEX .MIC [600,1204]       Index instruction      : INDEX

                                   ;10012  =0110   ;CALL CONSTRAINT BLOCK FOR MUL.S ROUTINE
                                   ;10013
                                   ;10014  INDEX.3:;0110-  ----------------;
U 0516, 0000,003D,0180,F800,0000,0430 ;10015       CALL,    J/MUL.S          ;  GO DO (INDEXIN+SUBSCRIPT)*SIZE
                                   ;10016
                                   ;10017          ;0111-------------------;  RETURN FROM MUL.S
                                   ;10018          ALU_D,  N&Z_ALU.V&C_0,   ;  SET CONDITION CODES FOR RESULT
U 0517, F001,003F,01F0,F847,0050,0300 ;10019       WRITE.DEST               ;  WRITE RESULT BY GOING TO C-FORK @ WRD:
                                   ;10020

                                   ;9421  MUL.S:  ;-----------------------;
                                   ;9422          R[R15]_Q, D_Q,          ;  SAVE M'CAND
                                   ;9423          SC_SC+K[SC],            ;  SC NOW CONTAINS 200
                                   ;9424          D.NE.0?                 ;  MUL'IER IS 0?
                                   ;9425
                                   ;9426  =101    ;0----------------------;
                                   ;9427          D_K[ZERO],N&Z_ALU.V&C_0,;  PROD IS 0 SINCE MUL'IER IS 0
                                   ;9428          RETURN8                 ;  WRITE RESULT 0
                                   ;9429
                                   ;9430          ;1----------------------;
                                   ;9431          Q_K[SC].CTX,            ;  SET SHF COUNT FOR B,W,L
                                   ;9432          LAB_R[R15]              ;  LATCH MUL'CAND
                                   ;9433  =;END
                                   ;9434          ;-----------------------;
                                   ;9435          SC_Q(EXP), STATE_Q(EXP),;  SC GETS COUNT (4,8,16) FOR B,W,L VIA EBMX
                                   ;9436          FE_Q(EXP), Q_D, DK/SHF, ;  SAVE CT TO REMEMBER B,W,L
                                   ;9437          RC[T0]_LB.LEFT,SI/ZERO  ;  RC 0 GETS 2 TIMES M'CAND
                                   ;9438
                                   ;9439  =0*     ;0----------------------;
                                   ;9440          D_RC[T1],Q_0,           ;  D GETS M'IER
                                   ;9441          STATE_STATE+FE,         ;  STATE HAS # BITS (8,16,32) FOR B,W,L
                                   ;9442          CALL, SIGNS?, J/MUL.6   ;  POS OR NEG MUL'CAND?
                                   ;9443
```