Bigmem

master acl2 / books / centaur / bigmem / bigmem.lisp

An ACL2 Model of a 2^{64}-byte Array

Shilpi Goel

Introduction

- Bigmem is based on the following paper:
  - Warren A. Hunt, Jr. and Matt Kaufmann. *A Formal Model of a Large Memory that Supports Efficient Execution*. FMCAD 2012.

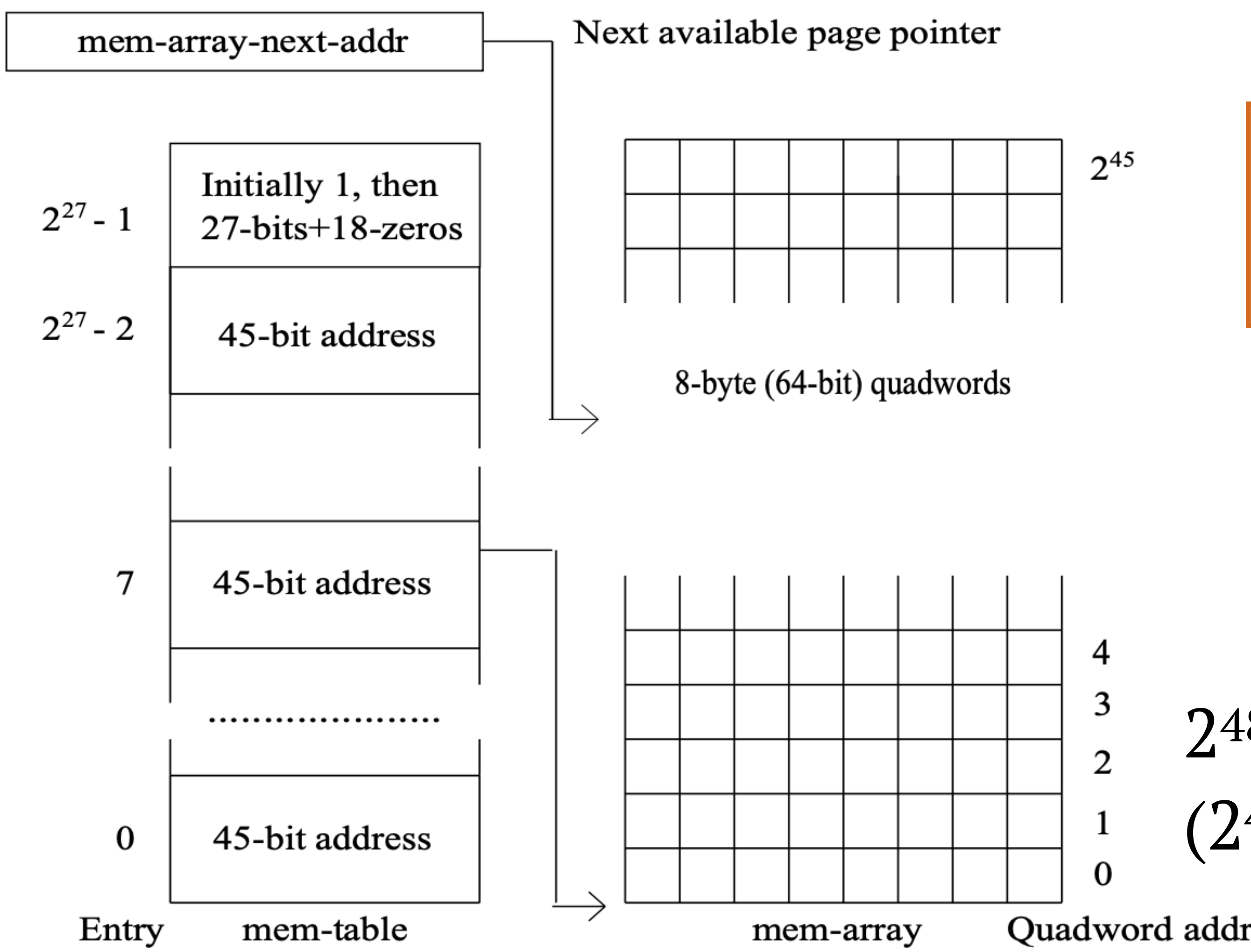


Fig. 1. Memory System

- Fair bit of arithmetic reasoning
- Not easy to tweak

2^{48}-byte memory
(2^{45} quadwords)

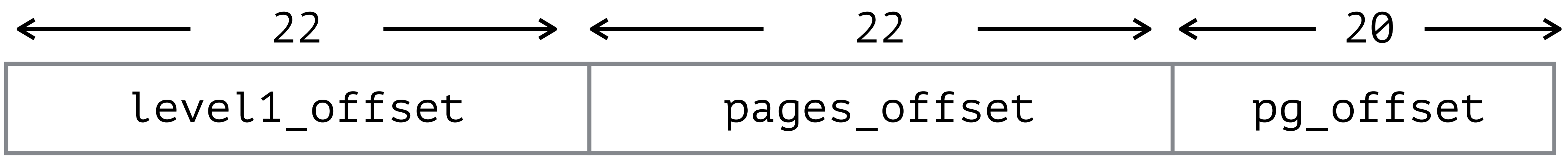
Bigmem: Exported Theorems

```
(defthm read-mem-over-write-mem
  (equal (read-mem addr-1 (write-mem addr-2 val mem))
         (if (equal addr-1 addr-2)
             (loghead 8 (ifix val))
           (read-mem addr-1 mem))))

(defthm write-mem-shadow-writes
  (equal (write-mem addr val-2 (write-mem addr val-1 mem))
         (write-mem addr val-2 mem)))

(defthm write-mem-commutes-safely
  (implies (not (equal addr-2 addr-1))
           (equal (write-mem addr-2 val-2 (write-mem addr-1 val-1 mem))
                  (write-mem addr-1 val-1 (write-mem addr-2 val-2 mem)))))

(defthm write-the-read
  (equal (write-mem addr (read-mem addr mem) mem)
         mem))

(defthm read-mem-from-nil
  (equal (read-mem i nil) 0))
```

Implementation

- Bigmem is implemented as an abstract stobj:
  - Concrete: a nest of resizable arrays; memory is allocated on demand
  - Abstract: a typed record; each element is a byte
- We will focus only on the concrete implementation here.

Concrete Data Structures



64-bit address

```
(defstobj page
  (pg :type (array (unsigned-byte 8) (0))
      :initially 0 :resizable t)
  (pg_vld :type bit :initially 0)
  :non-executable t)


(defstobj l1
  (pages :type (array page (0)) :resizable t)
  (pages_vld :type bit :initially 0)
  :non-executable t)


(defstobj mem$c
  (level1 :type (array l1 (*2^22*))
          :resizable nil))
```

```
(define write-to-page ((offset   :type (unsigned-byte 20))
                       (val      :type (unsigned-byte 8))
                       (page      good-pagep))
  (b* (...
       (page   (if (mbe
                    ;; Computing a resizable array's length could be
                    ;; a linear-time operation?
                    :logic (< offset (pg-length page))
                    ;; Reading pg_vld is always a constant-time operation.
                    :exec (equal (pg_vld page) 1))

                   page

                 (b* ((page (update-pg_vld 1 page))
                      (page (resize-pg *2^20* page)))
                   page)))

       (page (update-pgi offset val page)))

    page))


(define good-pagep (page)
  (and (pagep page)
       (if (equal (pg_vld page) 0)
           (equal (pg-length page) 0)
         (equal (pg-length page) *2^20*))))


(defthm write-to-page-shadow-writes
  (equal (write-to-page offset val2 (write-to-page offset val1 page))
         (write-to-page offset val2 page)))
```

```lisp
(define read-from-page ((offset :type (unsigned-byte 20))
                        (page   good-pagep))
   (if (mbe :logic (< offset (pg-length page))
            :exec (equal (pg_vld page) 1))
        (pgi offset page)
      ;; Default memory value
      0))


(defthm read-write-page
  (equal (read-from-page offset1 (write-to-page offset2 val page))
         (if (equal (loghead 20 offset1) (loghead 20 offset2))
             (loghead 8 val)
           (read-from-page offset1 page))))
```

For each data structure, define analogous read and write functions, and their corresponding theorems.

Space Usage

```
(defstobj page
  (pg :type (array (unsigned-byte 8) (0))
      :initially 0 :resizable t)
  (pg_vld :type bit :initially 0)
  :non-executable t)

(defstobj l1
  (pages :type (array page (0)) :resizable t)
  (pages_vld :type bit :initially 0)
  :non-executable t)

(defstobj mem$c
  (level1 :type (array l1 (*2^22*))
          :resizable nil))
```

- Initially, `mem$c` has 2^{22} `(create-l1)` elements.
- When a write occurs:
  - `pages` is resized to 2^{22} `(create-page)` elements.
  - 2^{20} bytes are allocated only for the `pg` in the selected `page`.
- Works well in the common scenario of spatial locality.

Conclusion

- Bigmem implementation is easily modifiable:
  - Can add more levels (e.g., `l2`, `l3`, etc.) if a larger memory is needed.
  - Can modify the maximum lengths of the arrays without tedious arithmetic reasoning.
- Execution overhead of using nested stobjs is almost negligible here.
- Bigmem is a general, reusable solution:
  - No familiarity needed with the underlying implementation.
  - E.g., can be used as a child stobj in the field of a parent stobj that models some machine's state (e.g., `x86isa` state).

ACL2::projects

Bigmem

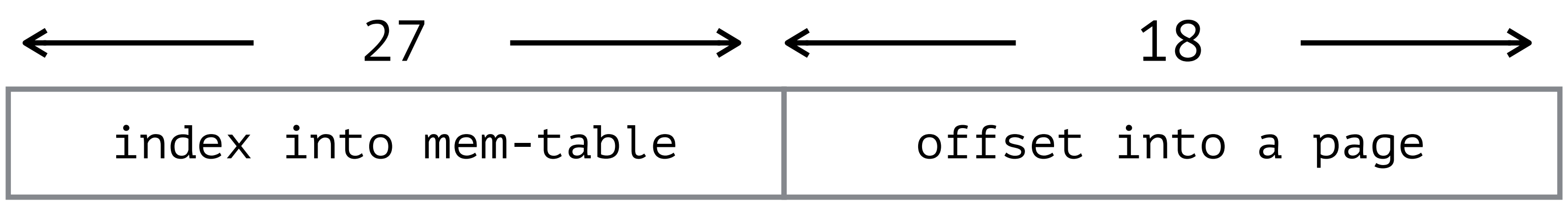
[books]/centaur/bigmem/bigmem.lisp

BIGMEM
Package

A `2^64`-byte memory model that is logically a record but provides array-like performance during execution

Thank You!

FMCAD'12 Paper: Worked Example



`45-bit quadword address`

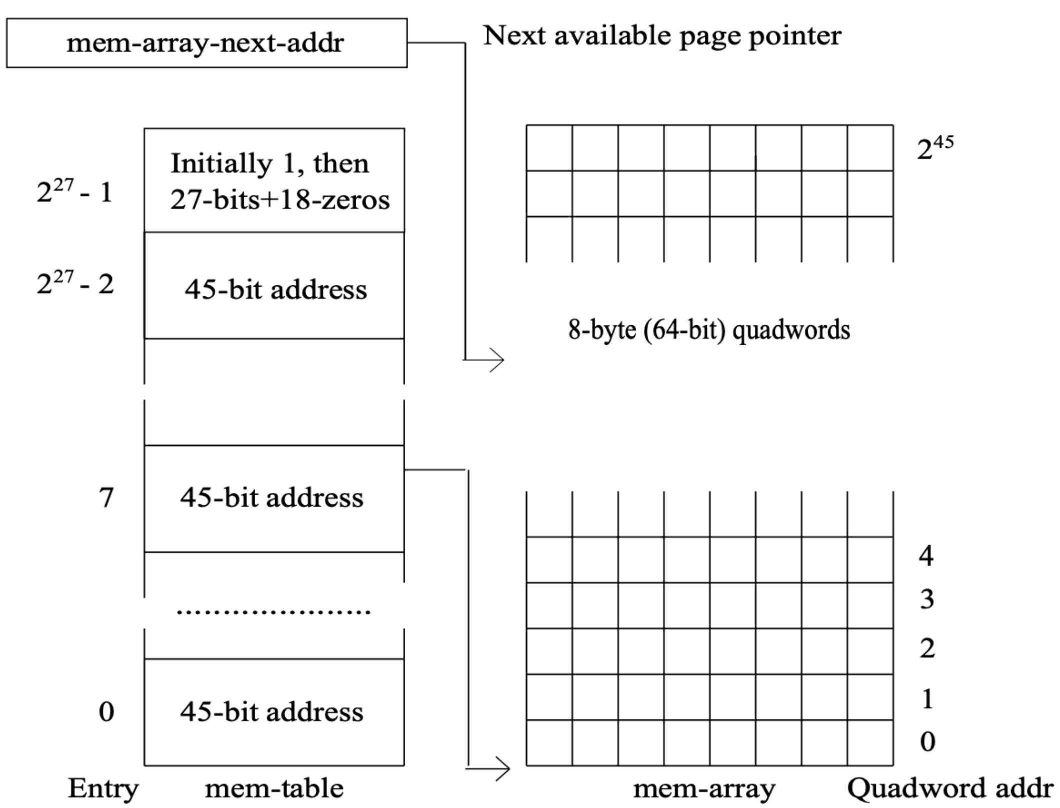


Fig. 1. Memory System

1. Write to quadword address
   `(7 * 2`^{18}`) + 345`
2. If `mem-table[7]` is valid, then
   page base address `=`
   `(mem-table[7] * 2`^{18}`)`
3. If `mem-table[7]` is invalid:
   a. page base address =
      `(mem-array-next-addr * 2`^{18}`)`
   b. `mem-array-next-addr =`
      `(mem-array-next-addr + 2`^{18}`)`
4. Final memory address =
   page base address + `345`