```lisp
(in-package "EMU")

(defun put-field (n a b)
  (let ((mask (- (expt 2 (+ (- a b) 1)) 1)))
    (ash (logand mask n) b)))

(defun make-opc (a b)
  (+ (put-field a 15 8) (put-field b 7 0)))

(setf *instruction-list*
      (mapcar #'(lambda ( w) (list (car w) (cadr w) (caddr w) (eval (cadddr w)))
)
              '(
        (%+ 2 msys::%+ (make-opc 0 32))
        (%- 2 msys::%- (make-opc 1 32))
        (%* 2 msys::%* (make-opc 100 32))

        (%div 2 msys::%div (make-opc 101 32))
        (%rem 2 msys::%rem (make-opc 102 32))

        (%<= 2 msys::%<= (make-opc 2 32))
        (%>= 2 msys::%>= (make-opc 3 32))
        (%< 2 msys::%< (make-opc 4 32))
        (%> 2 msys::%> (make-opc 5 32))
        (%= 2 msys::%= (make-opc 6 32))
        (%/= 2 msys::%/= (make-opc 13 32))

        (p-align3 1 p-align3-fun (make-opc 32 33))
        (p-constr 2 msys::p-constr (make-opc 37 32))

        (p-add 2 msys::p-add (make-opc 38 32))
        (p-shiftl-1 1 p-shiftl-fun (make-opc 39 33))
        (p-shiftr-1 1 p-shiftr-fun (make-opc 40 33))

        (tag-and 2 msys::tag-and (make-opc 16 32))
        (tag-ior 2 msys::tag-ior (make-opc 17 32))
        (tag-xor 2 msys::tag-xor (make-opc 18 32))

        (tag-eq 2 msys::tag-eq (make-opc 19 32))

        (get-tag 1 msys::get-tag (make-opc 20 32))
        (tag-set 2 msys::tag-set (make-opc 21 32))


        (addr-eq 2 msys::addr-eq (make-opc 24 32))
        (addr-eqi 2 msys::addr-eqi (make-opc 25 32))
        (addr-and 2 msys::addr-and (make-opc 26 32))
        (addr-ior 2 msys::addr-ior (make-opc 27 32))
        (get-addr 1 msys::get-addr (make-opc 28 32))

        (get-byte 2 msys::get-byte (make-opc 0 48))
        (set-byte 3 msys::set-byte (make-opc 0 49))

        (get-word 2 msys::get-word (make-opc 0 50))
        (set-word 3 msys::set-word (make-opc 0 51))

        (get-hword 2 msys::get-hword (make-opc 0 52))
        (set-hword 3 msys::set-hword (make-opc 0 53))

        (get-status 0 get-status-fun (make-opc 0 42))
        (set-status 1 set-status-fun (make-opc 0 43))

        (get-dsp 0 get-dsp-fun (make-opc 0 44))
        (get-csp 0 get-csp-fun (make-opc 0 45))
```



```lisp
        (get-a-index-pp 0 get-a-index-pp-fun (make-opc 0 56))
        (set-b-index-pp 1 set-b-index-pp-fun (make-opc 0 57))

        (set-a 1 set-a-fun (make-opc 0 58))
        (set-b 1 set-b-fun (make-opc 0 59))

        (consp 1 msys::consp (make-opc 255 0))

        (JNIL 0 jnil-fun (make-opc 0 11))
        (JMP 0 jmp-fun (make-opc 0 12))

        (LOD 0 lod-fun (make-opc 0 1))
        (LODNIL 0 lodnil-fun (make-opc 0 2))

        (STO 0 sto-fun (make-opc 0 3))

        (LITIDX 0 litidx-fun (make-opc 0 4))

        (INCSP 0 incsp-fun (make-opc 0 5))
        (DECSP 0 decsp-fun (make-opc 0 6))

        (HALT 0 halt-fun (make-opc 0 30))

        (CALLCLOS 0 callclos-fun (make-opc 0 16))
        (CALLCLOSV 0 callclosv-fun (make-opc 0 18))
        (RET 0 ret-fun (make-opc 0 20))

        (LODCS 0 lodcs-fun (make-opc 0 7))
        (STOCS 0 stocs-fun (make-opc 0 8))

        (PUSHCS 0 pushcs-fun (make-opc 0 9))
        (INCCSP 0 inccsp-fun (make-opc 0 10))

        (EXCEPT 0 except-fun (make-opc 0 31))

        (LITIDXSYMFUN 0 litidxsymfun-fun (make-opc 0 34))

        (%read 0 read-fun (make-opc 255 1))
        (%print 1 print-fun (make-opc 0 60))

        (%make-closure 2 msys::make-closure (make-opc 255 41))
        (%make-vector 1 msys::make-vector (make-opc 255 40))

        (%set-symbol-function 2 msys::set-symbol-function (make-opc 0 37))
        (%set-symbol-value 2 msys::set-symbol-value (make-opc 0 39))

        (%symbol-function 1 msys::symbol-function (make-opc 0 36))
        (%symbol-value 1 msys::symbol-value (make-opc 0 38))

        (%svref 2 msys::svref (make-opc 0 22))
        (%set-svref 3 msys::set-svref (make-opc 0 24))

        (%closure-ref 2 msys::closure-ref (make-opc 0 26))

        (eq 2 msys::eq (make-opc 255 12))
        (car 1 msys::car (make-opc 0 28))
        (cdr 1 msys::cdr (make-opc 0 29))
        (cons 2 msys::cons (make-opc 255 15)))))



(setf *num-instr* (length *instruction-list*))
```



```lisp
(setf *dispatch* (make-array *num-instr*))


; die opcode-liste

(setf *opcode-list* nil)
(setf *instr-list-len* (length *instruction-list*))

(setf *opcode-vector* (make-array *instr-list-len*))


(defun make-opcode-vector ()
  (do ((i 0 (+ i 1))
       (p *instruction-list* (cdr p)))
      ((>= i *instr-list-len*))
    (setf (aref *opcode-vector* i) (car p))))

(make-opcode-vector)

(setf *r-instr-assoc* (make-hash-table))

(defun make-instr-assoc ()
  (let ((res nil))
    (do ((i 0 (+ i 1)))
        ((>= i *instr-list-len*) res)
      (setf res (cons (cons (aref *opcode-vector* i) (nth 3 (aref *opcode-vector
* i))) res))
      (setf (gethash (nth 3 (aref *opcode-vector* i)) *r-instr-assoc*) (aref *op
code-vector* i)))))

(setf *instr-assoc* (make-instr-assoc))

(defun get-opcode (instr)
  (let ((subs (assoc-if #'(lambda (z) (eql (car z)  instr)) *instr-assoc*)))
    (if (null subs)
        (let ()
          (format t "instruction / ~A / not found~%" instr)
          (error "instruction not found"))
      (cdr subs))))



; der virtuelle prozessor

; initialisierung

(defun emu-init (tp-init)
; program counter
  (setf *pc* 0)

; data stack pointer
  (setf *dsp* 8192)

; control stack pointer
  (setf *csp* 4096)

  (setf *halt-flagged* nil)

; regs and status word

  (setf *regA* 0)
  (setf *regB* 0)

  (setf *status* 0)
```



```lisp

; template pointer
  (msys::set-word 252 0 tp-init)
  (setf *tp* (msys::get-word 252 0))

  (setf *pc* (msys::get-word *tp* (msys::cl-to 4))))



(setf *watch-instr* nil)

(defun debug-emu (on)
  (if on
      (let ()
        (setf *watch-instr* 1))
    (let ()
      (setf *watch-instr* nil))))


(defparameter *instr-mask* (ash (- (expt 2 16) 1) 16))
(defparameter *ext-mask* 65535)


(defun instr-opc (instr)
  (ash (logand instr *instr-mask*) -16))

(defun instr-ext (instr)
  (logand instr *ext-mask*))

(defun logicalp (x)
  (or (eq x t) (eq x nil)))

(defun get-opc-entry (opc)
  (let ((res (gethash opc *r-instr-assoc* nil)))
    (if (not (null res))
        res
      (error "can not get opc-entry"))))



(defun run-processor (max-steps)
  (let ((instr nil)
        (opc nil)
        (nargs nil)
        (args nil)
        (old-pc nil)
        (old-dsp nil)
        (old-csp nil)
        (opc-ext nil)
        (step-cnt nil))
    (do ((step-cnt 0 (+ 1 step-cnt)))
        ((or *halt-flagged* (>= step-cnt max-steps)) (format t "~%~% EMU-Steps = ~
A~%~%" step-cnt))
      (setf instr (msys::get-word *pc* 0))
      (setf opc (instr-opc instr))
;       (setf opc-entry (aref *opcode-vector* opc))
      (setf opc-entry (get-opc-entry opc))
      (setf opc-ext (instr-ext instr))
      (if *watch-instr*
          (format t "emu:: instr = ~x: ~A ~A~%" *pc* (car opc-entry) opc-ext))
      (setf nargs (cadr opc-entry))
      (setf args (let ((res nil))
                   (do ((i 0 (+ i 1)))
                       ((>= i nargs) res)
```

```
                     (setf res (cons (msys::get-word (+ *dsp* (* i 4)) 0) res)))
                   res))

      (setf old-pc *pc*)
      (setf old-dsp *dsp*)
      (setf old-csp *csp*)
      (setf *jmp-flag* 0)
      (if (> nargs 0)
          (let ()
            (setf res (apply (caddr opc-entry) args))
            (when (logicalp res)
;                (format t "umformung logisch:")
                 (setf res (msys::log-to res))))
        (setf res (apply (caddr opc-entry) (list instr))))
      (if (> nargs 0)
          (let ()
            (setf *dsp* (+ *dsp* (* 4 (- nargs 1))))
            (msys::set-word *dsp* 0 res)))
      (if (zerop *jmp-flag*)
          (setf *pc* (+ *pc* 4))))))



(defun push-stack (val)
  (let ()
    (setf *dsp* (- *dsp* 4))
    (msys::set-word *dsp* 0 val)))

(defun pop-stack ()
  (let ()
    (setf *dsp* (+ *dsp* 4))
    (msys::get-word (- *dsp* 4) 0)))

(defun push-address-stack (val)
  (let ()
    (setf *csp* (- *csp* 4))
    (msys::set-word *csp* 0 val)))

(defun pop-address-stack ()
  (let ()
    (setf *csp* (+ *csp* 4))
    (msys::get-word (- *csp* 4) 0)))

(defun litidx-fun (instr)
  (let ((val (msys::templ-ref-d *tp* (msys::cl-to (instr-ext instr)))))
    (push-stack val)))

(defun decsp-fun (instr)
  (let ((d (instr-ext instr)))
    (setf *dsp* (- *dsp* (* 4 d)))))

(defun incsp-fun (instr)
  (let ((d (instr-ext instr)))
    (setf *dsp* (+ *dsp* (* 4 d)))))

(defun inccsp-fun (instr)
  (let ((d (instr-ext instr)))
    (setf *csp* (+ *csp* (* 4 d)))))

(defun lod-fun (instr)
  (let ((d (instr-ext instr)))
    (let ((val (msys::get-word (+ *dsp* (* 4 d)) 0)))
      (push-stack val))))

(defun pushcs-fun (instr)
```



```
  (let ((d (instr-ext instr)))
    (let ((val (msys::get-word (+ *dsp* (* 4 d)) 0)))
      (push-address-stack val))))

(defun lodcs-fun (instr)
  (let ((d (instr-ext instr)))
    (let ((val (msys::get-word (+ *csp* (* 4 d)) 0)))
      (push-stack val))))

(defun stocs-fun (instr)
  (let ((d (instr-ext instr)))
    (let ((val (msys::get-word *dsp* 0)))
      (setf *dsp* (+ *dsp* 4))
      (msys::set-word (+ *csp* (* 4 d)) 0 val))))

(defun lodnil-fun (instr)
  (push-stack msys::nil))


(defun sto-fun (instr)
  (let ((d (instr-ext instr)))
    (let ((val (msys::get-word *dsp* 0)))
      (setf *dsp* (+ *dsp* 4))
      (msys::set-word (+ *dsp* (* 4 (- d 1))) 0 val))))


(defun two-comp (x)
  (if (>= x 32768)
      (- x 65536)
    x))


(defun jnil-fun (instr)
  (let ((d (instr-ext instr)))
    (let ((val (pop-stack)))
      (if (eq val msys::nil)
          (let ()
            (setf *pc* (+ *pc* (* 4 (two-comp d))))
            (setf *jmp-flag* 1))))))

(defun jmp-fun (instr)
  (let ((d (instr-ext instr)))
    (setf *pc* (+ *pc* (* 4 (two-comp d))))
    (setf *jmp-flag* 1)))

(defun halt-fun (instr)
  (setf *halt-flagged* t))

(defun read-fun (instr)
  )

(defun print-fun (val)
  (format t "EMU: ~A~%" (msys::to-cl val))
  val)

(defun except-fun (instr)
  (let ()
    (format t "*** EMU: exception occurred~%")
    (setf *halt-flagged* true)))


(defun callclos-fun (instr)
  (let ((d (instr-ext instr)))
;    (format t "d = ~A~%" d)
    (let ((clos (msys::get-word (+ *dsp* (* 4 (- d 1))) 0)))
```



```
      (push-stack (msys::cl-to d))
;       (format t "type clos= ~A~%" (msys::show-types clos))
      (let ((templ (msys::closure-ref clos 0)))
        (push-address-stack *tp*)
        (setf *tp* templ)
        (push-address-stack (+ *pc* 4))
        (setf *pc* (msys::get-word *tp* (msys::cl-to 4)))
        (setf *jmp-flag* 1)))))

(defun callclosv-fun (instr)
  (let ((d (msys::to-cl (msys::get-word *dsp* 0))))
;    (format t "d = ~A~%" d)
    (let ((clos (msys::get-word (+ *dsp* (* 4 d)) 0)))
;    (format t "type clos= ~A~%" (msys::to-cl clos))
      (let ((templ (msys::closure-ref clos 0)))
        (push-address-stack *tp*)
        (setf *tp* templ)
        (push-address-stack (+ *pc* 4))
        (setf *pc* (msys::get-word *tp* (msys::cl-to 4)))
        (setf *jmp-flag* 1)))))


(defun ret-fun (instr)
  (let ((new-pc (pop-address-stack))
        (new-tp (pop-address-stack)))
    (setf *tp* new-tp)
    (setf *pc* new-pc)
    (setf *jmp-flag* 1)))

(defun p-align3-fun (x)
  (msys::%align x (msys::cl-to 3)))

(defun p-shiftl-fun (x)
  (msys::p-shiftl x (msys::cl-to 1)))

(defun p-shiftr-fun (x)
  (msys::p-shiftr x (msys::cl-to 1)))


(defun get-a-index-pp-fun (instr)
  (push-stack (msys::get-word *regA* 0))
  (setf *regA* (+ *regA* 4)))

(defun set-b-index-pp-fun (x)
  (let ((res (msys::set-word *regB* 0 x)))
    (setf *regB* (+ *regB* 4))
    res))



(defun set-a-fun (p)
  (setf *regA* p))


(defun set-b-fun (p)
  (setf *regB* p))

(defun litidxsymfun-fun (instr)
  (let ((d (instr-ext instr)))
    (let ((val (msys::templ-ref-d *tp* (msys::cl-to (instr-ext instr)))))
      (setf val (msys::symbol-function val))
      (push-stack val))))

(defun get-dsp-fun (instr)
  (push-stack *dsp*))
```



```
(defun get-csp-fun (instr)
  (push-stack *csp*))

(defun get-status-fun (instr)
  (push-stack *status*))

(defun set-status-fun (x)
  (setf *status* x))
```