```
// Eigen Intermediate Code Syntax
// Derived from the Eigen Compiler Suite User Manual, the asm/lexer/parser/checker code
// and code examples
// Author: me@rochus-keller.ch, 2024

// This is a grammar of the Eigen Intermediate Code as generated by the current Eigen compilers.

// NOTE: preprocessing directives are not considered here (supported by parser, but never used
// in the examples)
// NOTE: the asm instruction is not considered

Program ::= { Section }

/////////////// Sections
Section ::=
        code_section
      | init_code_section
      | data_section
      | init_data_section
      | const_section
      | type_section

code_section ::= '.code' identifier
                 { '.duplicable' | '.replaceable' }
                 [loc] [type]
                 { code
                   | sym_decl // params, locals
                   | break_decl }

code ::= data_management
       | arithmetic
       | bit_manipulation
       | function_call
       | branching
       | special
       | trap
       | alias
       | req

init_code_section ::= '.initcode' identifier
                      { '.required' | '.duplicable' }
                      { code }

data_section ::= '.data' identifier
                 ['.alignment' unsigned_int ]
                 { alias | req | data_decl }

init_data_section ::= '.initdata' identifier
                      { code }

const_section ::= '.const' (identifier|string)
                  ['.duplicable'
                  | '.alignment' unsigned_int ]
                  { alias | req | def }

type_section ::= '.type' identifier { loc | type }

/////////////// Instructions
data_management ::= 'mov' RegMem ',' ImmRegAdrMem
```

```
                        | 'conv' RegMem ',' ImmRegAdrMem
                        | 'copy' Pointer ',' Pointer ',' ImmRegMem
                        | 'fill'   Pointer ',' Pointer ',' ImmRegAdrMem

arithmetic ::= 'add' RegMem ',' ImmRegAdrMem ',' ImmRegAdrMem
             | 'sub' RegMem ',' ImmRegAdrMem ',' ImmRegAdrMem
             | 'mul' RegMem ',' ImmRegMem ',' ImmRegMem
             | 'div'  RegMem ',' ImmRegMem ',' ImmRegMem
             | 'mod' RegMem ',' ImmRegMem ',' ImmRegMem
             | 'neg' RegMem ',' ImmRegMem

bit_manipulation ::= 'and' RegMem ',' ImmRegMem ',' ImmRegMem
                   | 'or'  RegMem ',' ImmRegMem ',' ImmRegMem
                   | 'xor' RegMem ',' ImmRegMem ',' ImmRegMem
                   | 'not' RegMem ',' ImmRegMem
                   | 'lsh' RegMem ',' ImmRegMem ',' ImmRegMem
                   | 'rsh' RegMem ',' ImmRegMem ',' ImmRegMem

function_call ::= 'push' ImmRegAdrMem
                | 'pop' RegMem
                | 'call' Function ',' Size
                | 'ret'
                | 'enter' Size
                | 'leave'

branching ::= 'br' Offset
            | 'jump' Function
            | 'breq' Offset ',' ImmRegAdrMem ',' ImmRegAdrMem
            | 'brne' Offset ',' ImmRegAdrMem ',' ImmRegAdrMem
            | 'brlt'   Offset ',' ImmRegAdrMem ',' ImmRegAdrMem
            | 'brge' Offset ',' ImmRegAdrMem ',' ImmRegAdrMem

special ::= 'nop'
          | 'fix'  Register
          | 'unfix' Register

loc ::= 'loc' string ',' Size ',' Size

alias ::= 'alias' string

req ::= 'req' string

trap ::= 'trap' Size

break_decl ::= 'break' loc

Offset ::= ('+'|'-') unsigned_int
Size ::= unsigned_int

/////////////// Type declarations
type ::=  'void'
        | typeref
        | array_decl
        | rec_decl
        | ptr_decl
        | ref_decl
        | func_decl
        | enum_decl
```

```
typeref ::= 'type' (string|basic_type)

rec_decl ::= 'rec' Offset ',' Size { field_decl }
field_decl ::= field loc type
field ::= 'field' string ',' Size ',' basic_type immediate_op

array_decl ::= 'array' Size ',' Size type

ptr_decl ::= 'ptr' type
ref_decl ::= 'ref' type

enum_decl ::= 'enum' Offset typeref { enum_elem }
enum_elem ::= value loc
value ::= 'value' string ',' basic_type immediate_op

func_decl ::= 'func' Offset type { type } // return type, followed by arg types

basic_type ::= 's1' | 's2' | 's4' | 's8' |
               'u1' | 'u2' | 'u4' | 'u8' |
               'f4' | 'f8' |
               'ptr' | 'fun'

/////////////// Data declarations
data_decl ::= loc type { def | res }
def ::= 'def' ImmAdr
res ::= 'res' Size

sym_decl ::= 'sym' Offset ',' string ',' ImmRegMem
             loc type

register_ ::= '$0' | '$1' | '$2' | '$3' | '$4' | '$5' | '$6' | '$7' |
              '$res' | '$sp' | '$fp' | '$lnk'

/////////////// Operators
immediate_op ::= ['+'|'-'] number
register_op ::= register_ [ ('+'|'-') number ]
address_op ::= '@' (identifier | string) [
                 \LL:2\ ('+'|'-') number
                 | '+' register_ [ ('+'|'-') number ] ]
memory_op ::= '[' ( number
                 | register_ [('+'|'-') number]
                 | '@' (identifier | string) [
                    \LL:2\ ('+'|'-') number
                    | '+' register_ [ ('+'|'-') number ] ] ) ']'

ImmAdr ::= basic_type (immediate_op | address_op)
RegMem ::= basic_type (register_op | memory_op)
ImmRegMem ::= basic_type (immediate_op | register_op | memory_op)
ImmRegAdrMem ::= basic_type (immediate_op | register_op | address_op | memory_op)
Pointer ::= 'ptr' (immediate_op | register_op | address_op | memory_op)
Function ::= 'fun' (immediate_op | register_op | address_op | memory_op)
Register ::= basic_type register_

number ::= unsigned_int | real
unsigned_int ::= decimal_integer | binary_integer | octal_integer | hex_integer

/////////////// Lexer tokens
string ::=          // '"' {letter} '"'
identifier ::=      // { letter | '.' | '$' | '_' | ':' }+
```

```
binary_integer ::=    // '0b' bin_digits | bin_digits 'b'
octal_integer ::=     // '0o' oct_digits | oct_digits 'o' | '0' oct_digits
decimal_integer ::= // '0d' dec_digits | dec_digits 'd' | dec_digits
hex_integer ::=       // '0h' hex_digits | hex_digits 'h' | '0x' hex_digits
real ::=              // dec_digits '.' [ dec_digits ] [ 'e' ['+'|'-'] dec_digits ]

comment- ::= ';'
Comment ::=

/// Pragmas
%namespace ::= 'lr'
```