Lýsing:

```
EBNF skilgreining ámálinu nema að OP hefur verið einfaldað
þannig að allar tvíundaraðgerðir hafi sama forgang en minni
forgang en einundaraðgerðir.
=============================================================
program = { function }
    ;

func
    : 'fun' NAME '(' [ NAME, { ',', NAME } ], ')', '{', {decl}, {body}, '}'
    ;

decl
    : 'var' NAME, {',', NAME}
    ;

expr
    : NAME '=' expr
    | binopexpr
    ;

binopexpr
    : binopexpr OP optmp1


smallexpr
    : NAME
    | NAME '(' [ expr, { ',', expr } ], ')'
    | OP, smallexpr
    | LITERAL
    | '(', expr, ')'
    ;

ifexpr
    : '(', expr, ')', '{', body, '}', iftail
    ;

iftail
    : 'elseif', '(', expr, ')', '{', body, '}', elseifexpr
    | 'else', '{', body, '}'
    ;

body
    : expr, ';', body
    | 'while', '(', expr, ')', '{', body, '}', body
    | 'if', ifexpr, body
    | 'return', expr, ';', body
    |
    ;
```

Lesgreinirinn:

```
%%
%public
%class NanomorphoLexer
%byaccj

%unicode

%{

public Borpho yyparser;

public NanomorphoLexer( java.io.Reader r, Borpho yyparser )
{
    this(r);
    this.yyparser = yyparser;
}

%}

NEWLINE    =   \n
WS         =   [ \t\r\f]
INTEGER    =   [0-9]+
FLOAT      =   [0-9]+\.[0-9]+([Ee][+-]?[0-9]+)?
BOOLEAN    =   (true)|(false)
NAME       =   [a-zA-Z_][a-zA-Z0-9_]*
OP1        =   [*/%]
OP2        =   [+-]
OP3        =   (<)|(>)|(>=)|(<=)|(==)
OP4        =   (\|\|)|(&&)
OP5        =   [-+/*%<>=|!&]+
DELIMITERS =   [()\{\};.,]
STRING     =   \"([^\"\n\r\f])*\"

COMMENT    =   ;;;.*

%%

{NEWLINE}
{
//  System.out.println("Newline!");
}
{COMMENT}
{
//  System.out.println("Comment!");
}
{WS}
{
//  System.out.println("Whitespace!");
}

var
{
    yyparser.yylval = new BorphoVal(yytext());
    return yyparser.VAR;
}

if
```

```
{
    yyparser.yylval = new BorphoVal(yytext());
    return yyparser.IF;
}

else
{
    yyparser.yylval = new BorphoVal(yytext());
    return yyparser.ELSE;
}

elseif
{
    yyparser.yylval = new BorphoVal(yytext());
    return yyparser.ELSEIF;
}

while
{
    yyparser.yylval = new BorphoVal(yytext());
    return yyparser.WHILE;
}

return
{
    yyparser.yylval = new BorphoVal(yytext());
    return yyparser.RETURN;
}

fun
{
    yyparser.yylval = new BorphoVal(yytext());
    return yyparser.FUN;
}

{DELIMITERS}
{
    yyparser.yylval = new BorphoVal(yytext());
    return (int) yycharat(0);
}

=
{
    yyparser.yylval = new BorphoVal(yytext());
    return yyparser.ASSIGNMENT;
}

{OP1}
{
    yyparser.yylval = new BorphoVal(yytext());
    return yyparser.OP1;
}

{OP2}
{
    yyparser.yylval = new BorphoVal(yytext());
    return yyparser.OP2;
}
```

```
116
117 {OP3}
118 {
119     yyparser.yylval = new BorphoVal(yytext());
120     return yyparser.OP3;
121 }
122
123 {OP4}
124 {
125     yyparser.yylval = new BorphoVal(yytext());
126     return yyparser.OP4;
127 }
128
129 {OP5}
130 {
131     yyparser.yylval = new BorphoVal(yytext());
132     return yyparser.OP5;
133 }
134
135 {INTEGER}
136 {
137     yyparser.yylval = new BorphoVal(yytext());
138     return yyparser.LITERAL;
139 }
140
141 {FLOAT}
142 {
143     yyparser.yylval = new BorphoVal(yytext());
144     return yyparser.LITERAL;
145 }
146
147 {BOOLEAN}
148 {
149     yyparser.yylval = new BorphoVal(yytext());
150     return yyparser.LITERAL;
151 }
152
153 {STRING}
154 {
155     yyparser.yylval = new BorphoVal(yytext());
156     return yyparser.LITERAL;
157 }
158
159 {NAME}
160 {
161     yyparser.yylval = new BorphoVal(yytext());
162     return yyparser.NAME;
163 }
164
165 .
166 {
167     System.out.println("Oh no. Token \"" + yytext() + "\" not know.");
168     return Borpho.YYERRCODE;
169 }
```

Þáttarinn, milliþulusmiðurinn og þulusmiðurinn:

```
%{
    import java.io.*;
    import java.util.*;
%}

%token <sval> IF, ELSE, ELSEIF, VAR, WHILE, NAME, RETURN, FUN, OP1, OP2, OP3, OP4, OP5,
    LITERAL, ASSIGNMENT
%type <obj> binopexpr, optmp1, optmp2, optmp3, optmp4, expr, smallexpr
%%

start
    : program
    ;

program
    : {millithula.add("~FUN");} func
    | program {millithula.add("~FUN");} func
    ;

func
    : FUN NAME {millithula.add($2); args_counter = 0;} '(' optnames ')'
        {function_map.put($2, args_counter); millithula.add("~" + args_counter);} '{'
        decls {millithula.add("~ENDDECL");} body '}' {millithula.add("~ENDFUN");}
    ;

optnames
    : names
    |
    ;

decls
    : decls decl ';'
    |
    ;

decl
    : VAR names
    ;

names
    : NAME            {millithula.add($1); args_counter++;}
    | names ',' NAME  {millithula.add($3); args_counter++;}
    ;

expr
    : NAME ASSIGNMENT expr   {millithula.add("~ASSIGN"); millithula.add($1);
        millithula.add($3);}
    | binopexpr              {$$ = $1;}
    ;

binopexpr
    : binopexpr OP5 optmp1
    {
        ArrayList<Object> ret = new ArrayList<Object>();
        ret.add("~CALL");
        ret.add($2);
        ret.add($1);
```

```
54          ret.add($3);
55          ret.add("~ENDCALL");
56          $$ = ret;
57      }
58      | optmp1
59      ;
60
61  optmp1
62      : optmp1 OP4 optmp2
63      {
64          ArrayList<Object> ret = new ArrayList<Object>();
65          ret.add("~CALL");
66          ret.add($2);
67          ret.add($1);
68          ret.add($3);
69          ret.add("~ENDCALL");
70          $$ = ret;
71      }
72      | optmp2
73      ;
74
75  optmp2
76      : optmp2 OP3 optmp3
77      {
78          ArrayList<Object> ret = new ArrayList<Object>();
79          ret.add("~CALL");
80          ret.add($2);
81          ret.add($1);
82          ret.add($3);
83          ret.add("~ENDCALL");
84          $$ = ret;
85      }
86      | optmp3
87      ;
88
89  optmp3
90      : optmp3 OP2 optmp4
91      {
92          ArrayList<Object> ret = new ArrayList<Object>();
93          ret.add("~CALL");
94          ret.add($2);
95          ret.add($1);
96          ret.add($3);
97          ret.add("~ENDCALL");
98          $$ = ret;
99      }
100     | optmp4
101     ;
102
103 optmp4
104     : optmp4 OP1 smallexpr
105     {
106         ArrayList<Object> ret = new ArrayList<Object>();
107         ret.add("~CALL");
108         ret.add($2);
109         ret.add($1);
110         ret.add($3);
111         ret.add("~ENDCALL");
```

```
112         $$ = ret;
113     }
114     | smallexpr
115     ;
116 
117 smallexpr
118     : NAME {$$ = $1;}
119     | NAME '(' optexprs ')'
120     {
121         expr_tmp.add("~ENDCALL");
122         ArrayList<Object> ret = new ArrayList<Object>();
123         ret.add("~CALL");
124         ret.add($1);
125         int n = expr_tmp.size();
126         for (int i = 0; i < n; i++)
127         {
128             ret.add(expr_tmp.get(i));
129         }
130         expr_tmp.clear();
131         $$ = ret;
132     }
133     | OP5 smallexpr
134     {
135         ArrayList<Object> ret = new ArrayList<Object>();
136         ret.add("~CALL");
137         ret.add($1);
138         ret.add($2);
139         ret.add("~ENDCALL");
140         $$ = ret;
141     }
142     | OP4 smallexpr
143     {
144         ArrayList<Object> ret = new ArrayList<Object>();
145         ret.add("~CALL");
146         ret.add($1);
147         ret.add($2);
148         ret.add("~ENDCALL");
149         $$ = ret;
150     }
151     | OP3 smallexpr
152     {
153         ArrayList<Object> ret = new ArrayList<Object>();
154         ret.add("~CALL");
155         ret.add($1);
156         ret.add($2);
157         ret.add("~ENDCALL");
158         $$ = ret;
159     }
160     | OP2 smallexpr
161     {
162         ArrayList<Object> ret = new ArrayList<Object>();
163         ret.add("~CALL");
164         ret.add($1);
165         ret.add($2);
166         ret.add("~ENDCALL");
167         $$ = ret;
168     }
169     | OP1 smallexpr
```

```
170     {
171         ArrayList<Object> ret = new ArrayList<Object>();
172         ret.add("~CALL");
173         ret.add($1);
174         ret.add($2);
175         ret.add("~ENDCALL");
176         $$ = ret;
177     }
178     | LITERAL {$$ = $1;}
179     | '(' expr ')' {$$ = $2;}
180     ;
181
182 optexprs
183     : expr {expr_tmp.add($1);} moreexpr
184     |
185     ;
186
187 moreexpr
188     : ',' expr {expr_tmp.add($2);} moreexpr
189     |
190     ;
191
192 ifexpr
193     : '(' expr {millithula.add($2);} ')' '{' body '}' {millithula.add("~ENDIF");}
          elseifexpr
194     ;
195
196 elseifexpr
197     : ELSEIF {millithula.add("~ELSEIF");} '(' expr ')' {millithula.add($4);} '{' body '}'
          {millithula.add("~ENDELSEIF");} elseifexpr
198     | elseexpr
199     ;
200
201 elseexpr
202     : ELSE {millithula.add("~ELSE");} '{' body '}' {millithula.add("~ENDELSE");}
203     |
204     ;
205
206 body
207     : expr ';' {millithula.add($1);} body
208     | WHILE '(' expr ')' {millithula.add("~WHILE"); millithula.add($3);} '{' body '}'
          {millithula.add("~ENDWHILE");} body
209     | IF {millithula.add("~IF");} ifexpr body
210     | RETURN {millithula.add("~RETURN");} expr ';' {millithula.add($3);} body
211     |
212     ;
213
214 %%
215
216     private NanomorphoLexer lexer;
217
218     private int yylex()
219     {
220         int yyl_return = -1;
221         try
222         {
223             yylval = new BorphoVal(0);
224             yyl_return = lexer.yylex();
```

```
225         }
226         catch (IOException e)
227         {
228             System.err.println("IO error: " + e);
229         }
230         return yyl_return;
231     }
232 
233     public void yyerror(String error)
234     {
235         System.err.println("Error: " + error);
236     }
237 
238     public Borpho(Reader r)
239     {
240         lexer = new NanomorphoLexer(r,this);
241     }
242 
243     public static void thula(ArrayList<Object> millithula)
244     {
245         int i = 0;
246         while (true)
247         {
248             String millithulu_takn = (String)millithula.get(i);
249 
250             if (millithulu_takn.equals("~FUN"))
251             {
252                 var_counter = 0;
253                 variable_map.clear();
254 
255                 i++;
256                 masm.print("#\"");
257                 masm.print(millithula.get(i));
258                 masm.print("[f");
259 
260                 i++;
261                 millithulu_takn = (String)millithula.get(i);
262                 String tmp_str = "";
263                 while (((String)millithula.get(i)).charAt(0) != '~')
264                 {
265                     if (variable_map.get(millithulu_takn) != null) throw_error("Variable
                         \"" + millithulu_takn + "\" is already declared.");
266                     variable_map.put(millithulu_takn, (Integer)var_counter);
267                     var_counter++;
268                     millithulu_takn = (String)millithula.get(++i);
269                 }
270                 masm.print(((String)millithula.get(i++)).substring(1,
                     ((String)millithula.get(i - 1)).length()));
271                 masm.println("]\" = ");
272                 masm.println("[");
273                 masm.println("(MakeVal null)");
274                 millithulu_takn = (String)millithula.get(i);
275                 {
276                     masm.println(tmp_str);
277                     millithulu_takn = (String)millithula.get(i++);
278                     while (!millithulu_takn.equals("~ENDDECL"))
279                     {
280                         if (variable_map.get(millithulu_takn) != null)
```

```
                        throw_error("Variable \"" + millithulu_takn + "\" is already
                        declared.");
281                     variable_map.put(millithulu_takn, (Integer)var_counter);
282                     masm.println("(Push)");
283                     var_counter++;
284                     millithulu_takn = (String)millithula.get(i++);
285                 }
286                 i = thula_body(i, "~ENDFUN");
287             }
288             masm.println("];\n");
289         }
290         else if(millithulu_takn.equals("~END"))
291         {
292             return;
293         }
294         else
295         {
296             throw_error("Expected a function declaration.");
297         }
298     }
299 }
300
301 public static int thula_body(int i, String exit)
302 {
303     while (true)
304     {
305         Object check = millithula.get(i);
306         if (check instanceof ArrayList)
307         {
308             thula_call((ArrayList)millithula.get(i));
309             i++;
310             continue;
311         }
312         else if (check instanceof String)
313         {
314             String millithulu_takn = (String)millithula.get(i++);
315             if (millithulu_takn.equals("~ASSIGN"))
316             {
317                 String name = (String)millithula.get(i++);
318                 thula_expr(millithula.get(i));
319                 i++;
320                 Integer var_loc = variable_map.get(name);
321                 if (var_loc == null) throw_error("Can't assign unknown variable \"" +
                        name + "\"");
322                 masm.println("(Store " + var_loc + ")");
323                 continue;
324             }
325             if (millithulu_takn.equals("~IF"))
326             {
327                 int bottom_label = name_counter++;
328                 thula_expr(millithula.get(i));
329                 i++;
330                 masm.println("(GoFalse _L" + (name_counter) + ")");
331                 i = thula_body(i, "~ENDIF");
332                 masm.println("(Go _L" + bottom_label + ")");
333                 masm.println("_L" + (name_counter++) + ":");
334                 if (millithula.get(i) instanceof String)
335                 while (millithula.get(i).equals("~ELSEIF"))
```

```
336                 {
337                     int label = name_counter;
338 
339                     i++;
340                     thula_expr(millithula.get(i));
341                     i++;
342                     masm.println("(GoFalse _L" + label + ")");
343                     i = thula_body(i, "~ENDELSEIF");
344                     masm.println("(Go _L" + bottom_label + ")");
345                     masm.println("_L" + label + ":");
346                     name_counter++;
347                 }
348                 if (millithula.get(i) instanceof String)
349                 if (millithula.get(i).equals("~ELSE"))
350                 {
351                     i++;
352                     i = thula_body(i, "~ENDELSE");
353                 }
354                 masm.println("_L" + bottom_label + ":");
355 
356                 continue;
357             }
358             if (millithulu_takn.equals("~WHILE"))
359             {
360                 int label1 = name_counter;
361                 int label2 = name_counter + 1;
362                 name_counter += 2;
363 
364                 masm.println("_L" + label1 + ":");
365                 thula_expr(millithula.get(i));
366                 masm.println("(GoFalse _L" + (label2) + ")");
367                 i = thula_body(i, "~ENDWHILE");
368                 masm.println("(MakeVal 1)");
369                 masm.println("(Go _L" + label1 + ")");
370                 masm.println("_L" + label2 + ":");
371 
372                 continue;
373             }
374             if (millithulu_takn.equals("~RETURN"))
375             {
376                 thula_expr(millithula.get(i));
377                 i++;
378                 masm.println("(Return)");
379                 continue;
380             }
381             if (millithulu_takn.equals(exit))
382             {
383                 return i;
384             }
385             else
386             {
387                 continue;
388             }
389         }
390         else
391         {
392             throw_error("Weird error :(");
393         }
```

```
394             return i;
395         }
396     }
397
398     public static void thula_expr(Object expr)
399     {
400         if (!(expr instanceof String))
401         {
402             thula_call((ArrayList<Object>)expr);
403         }
404         else
405         {
406             if (is_literal((String)expr))
407             {
408                 masm.println("(MakeVal " + expr + ")");
409             }
410             else
411             {
412                 Integer var_loc = variable_map.get(expr);
413                 if (var_loc == null) throw_error("Undefined variable: " + expr);
414
415                 masm.println("(Fetch " + var_loc + ")");
416             }
417         }
418     }
419
420     public static void thula_call(ArrayList call)
421     {
422         int i = 1;
423         String name = (String)call.get(1);
424
425         if (call.size() != 3)
426         {
427             while (true)
428             {
429                 i++;
430                 if ((call.get(i + 1) instanceof String) && ((String)call.get(i +
                        1)).equals("~ENDCALL"))
431                 {
432                     thula_expr(call.get(i));
433                     break;
434                 }
435                 else
436                 {
437                     thula_expr(call.get(i));
438                     masm.println("(Push)");
439                 }
440             }
441             masm.println("(Call #\"" + name + "[f" + (i - 1) + "]\" " + (i - 1) + ")");
442         }
443         else
444         {
445             masm.println("(Call #\"" + name + "[f0]\" 0)");
446         }
447     }
448
449     public static boolean is_literal(String str)
450     {
```

```
451         return is_double(str) || is_string(str) || is_boolean(str) || is_null(str);
452     }
453     public static boolean is_string(String str)
454     {
455         return str.charAt(0) == '\"';
456     }
457     public static boolean is_boolean(String str)
458     {
459         return str.equals("true") || str.equals("false");
460     }
461     public static boolean is_null(String str)
462     {
463         return str.equals("null");
464     }
465     public static boolean is_double(String str)
466     {
467         try
468         {
469             double d = Double.parseDouble(str);
470             return true;
471         }
472         catch (NumberFormatException ex)
473         {
474             return false;
475         }
476     }
477 
478     public static void clean_millithula()
479     {
480         ArrayList<Object> tmp = new ArrayList<Object>();
481         int n = millithula.size();
482         for (int i = 0; i < n; i++)
483             if (millithula.get(i) != null) tmp.add(millithula.get(i));
484 
485         millithula = tmp;
486     }
487 
488 
489 
490 
491     public static HashMap<String, Integer> function_map;
492     public static HashMap<String, Integer> variable_map;
493     public static ArrayList<Object> millithula;
494     public static ArrayList<Object> expr_tmp;
495     public static PrintStream masm;
496 
497     public static int args_counter;
498     public static int name_counter;
499     public static int var_counter;
500 
501     public static void main(String[] args)
502         throws FileNotFoundException
503     {
504         Borpho yyparser = new Borpho(new FileReader(args[0]));
505         if (args.length == 0)
506         {
507             System.err.println("Fatal error. No input file.");
508             System.exit(1);
```

```
509         }
510
511         function_map = new HashMap<String, Integer>();
512         variable_map = new HashMap<String, Integer>();
513         millithula  = new ArrayList<Object>();
514         expr_tmp    = new ArrayList<Object>();
515         masm        = new PrintStream(new File(args[0] + ".masm"));
516         name_counter = 0;
517         var_counter = 1;
518
519         System.out.println("Parsing!");
520         yyparser.yyparse();
521         millithula.add("~END");
522         System.out.println("Parser finsished!");
523
524         clean_millithula();
525         print_millithula();
526
527         System.out.println();
528
529         masm.println("\"" + args[0] + ".mexe\" = main in\n!\n{{");
530         thula(millithula);
531         masm.println("}}\n*\nBASIS\n;");
532
533         System.out.println("Comilation complete!");
534     }
535
536     public static void print_millithula()
537     {
538         int n = millithula.size();
539
540         System.out.println("Printing the milliþula:");
541         for (int i = 0; i < n; i++)
542         {
543             System.out.println(millithula.get(i));
544         }
545     }
546
547     public static void throw_error(String error)
548     {
549         System.out.println(error);
550         System.exit(4);
551     }
```