SIMDC12 Parser

February 14, 2012

This document summarizes the second phase of the first compiler project: construction of the parser for SIMDC'08. You should be able to reuse your lexer and symbol table code with only minor editing.

1 SIMDC12 Language Syntax

SIMDC12 is based on a C subset extended for parallelism. Consequently, it follows C's syntax and semantics fairly closely. The following extended BNF grammar, combined with the same lexical conventions used by C, define the language structure.

```
prog: (decl)* (func)* EOF
    ;
func: WORD {':' NUMBER} '(' typ WORD (',' typ WORD)* ')' stat
    ;
decl: typ {'[' NUMBER ']'} WORD (',' WORD)* ';'
    ;
typ: ("mono" | "poly") {':' NUMBER}
   ;
stat: '{' (decl)* (stat)* '}'
    | "if" expr stat
    | "where" expr stat
    | "while" expr stat
    | "return" expr ';'
    | expr ';'
    | ';'
    ;
expr: expr1 {'='  expr}
    ;
expr1: expr2 ('|' expr2)*
     ;
expr2: expr3 ('^' expr3)*
     ;
expr3: expr4 ('&' expr4)*
     ;
```

```
expr4: expr5 (('<' | '>') expr5)*
     ;
expr5: expr6 (('+' | '-') expr6)*
     ;
expr6: expr7 ('*' expr7)*
     ;
expr7: expr8 {':' NUMBER}
     ;
expr8:
     | WORD {qual}
     | '-' expr8
     | '!' expr8
     | '~' expr8
     | '(' expr ')'
     | “NPROC”
     | “IPROC”
     ;
qual: '(' {expr (',' expr)*} ')'
    | '[' expr ']'
    | '.' NUMBER
    ;
```

2 SIMDC12 Symbol Table

The SIMDC dialect described above allows declarations to appear within braces, implementing the usual nested lexical scoping rules. You will need to make your symbol table appropriately deal with this. Further, for local variables, you will need to keep track of the position of each variable in the stack frame – the value associated with a local variable should be its offset in the stack frame for the current function.

You also need to keep track of `mono` vs. `poly`, the bit precision, and the array size for each variable. For example, `poly:4[5] i;` declares `i` to be an array of five 4-bit integers with potentially different values in each PE. Without a `:`, all values are assumed to have 8 bits. Declaring without a `:` yields 8-bit precision and unary suffix `:` can be used as an rvalue precision cast.

3 Error Handling

For this project, you are to output warning/error messages to `stderr`. Do not worry about errors and warnings in general; you may assume that the input is correct except in the following two ways:

1. In the above syntax, there are two places in which ',' occurs. In each of these cases, ',' should be treated as optional, producing a warning message like “`5: warning: missing ',' assumed`” and continuing to process

the rest of the input as if nothing was wrong. The line number (5 in the example) need not be precisely correct.

2. Issue a warning like "`5: warning: this precision may be expensive`" each time a variable with more than 12 bits is used.

3. In each case where a `WORD` appears in the grammar, your parser should issue an appropriate error message if the `WORD` has already been defined in the current scope or if the `WORD` is being used in an expression and has not been defined. An appropriate error message would thus be something like "`5: error: conflicting definition of yuck`" or "`5: error: yuck has not been defined`" – assuming the `WORD` is `yuck`. Your parser may exit after reporting an error.

4 Output

Your parser is going to function as a simple "pretty printer" for the input SIMDC12 program generating preformatted HTML output to `stdout`. Here are the formatting rules:

- Your complete output should start with `<PRE>` and end with `</PRE>`
- Every lexeme should output itself except `&` becomes `&amp;` , `<` becomes `&lt;` , and `>` becomes `&gt;`
- Every keyword should be made strong, e.g., `return` becomes `<STRONG>return</STRONG>`
- Every `mono` variable should be made red, e.g., `yuk` becomes `<FONT COLOR=RED>yuk</FONT>`
- Every function name or `poly` variable should be made green, e.g., `yuk` becomes `<FONT COLOR=GREEN>yuk</FONT>`
- Additionally, every local variable should be emphasized, e.g., local `poly yuk` becomes `<FONT COLOR=GREEN><EM>yuk</EM></FONT>`
- Every declaration (`decl`), function definition, or statement (`stat`) should start on a new line
- A stat involving braces changes indentation; the `{` and `}` should each be on their own lines, and identation of each line *within* should equal twice the number of enclosing braces
- After the indentation, you may use whatever spacing rules you wish within expressions and statements and may also insert blank lines

5 Submission

See the course WWW site, `http://aggregate.org/STCH/`, for details.