<<<< USERNAME:tt FILENAME: maspar.y >>>>

```
%{
//-----------------------------------------------------------------------
//File maspar.y contains a specification for MARIE Assembler Language
//defined by Thomas R. Turner.
//-----------------------------------------------------------------------
//Author:    Thomas R. Turner
//E-Mail:    trturner@uco.edu
//Date:      August, 2016
//-----------------------------------------------------------------------
//Copyright August, 2016 by Thomas R. Turner.
//Do not reproduce without permission from Thomas R. Turner.
//-----------------------------------------------------------------------
//C++ inlcude files
//-----------------------------------------------------------------------
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <cstdio>
using namespace std;
//-----------------------------------------------------------------------
//-----------------------------------------------------------------------
//Application inlcude files
//-----------------------------------------------------------------------
#include "maslex.h"
#include "maspar.h"
#include "masopcodes.h"
#include "masutilities.h"
#include "maslist.h"
#include "maslabel.h"
#include "masfiles.h"
//-----------------------------------------------------------------------
//Functions
//-----------------------------------------------------------------------
void yyerror(const char* m);
void MemoryPrint(ostream& o);
void MemoryTitle(ostream& o);
void MemoryWrite(FILE* f);
//-----------------------------------------------------------------------
//Externals
//-----------------------------------------------------------------------
extern Files F;                              //Files specific to the MARIE
                                             //Assembler

extern int Line;
extern int Col;
//-----------------------------------------------------------------------
//Global Variables
//-----------------------------------------------------------------------
unsigned short int address=0;                //Address of the next instruction
unsigned short int origin=0;                 //Address of the first instruction
unsigned short int length=0;                 //Number of words (16-bit) stored
                                             //in MARIE's memory
unsigned short int Memory[4096];             //MARIE's memory
Label LT;                                    //Label Table for resolving the addresses
 of labels
//-----------------------------------------------------------------------
%}
%union {
  string* token;
  unsigned short integer;
}
%type  <integer> operand
%token <token>   TOKEN_BEGIN
%token <token>   COMMA
%token <token>   RESERVE_WORDS
```

<<<< USERNAME:tt FILENAME: maspar.y >>>>

```
%token <token>    LOAD
%token <token>    STORE
%token <token>    ADD
%token <token>    SUBT
%token <token>    INPUT
%token <token>    OUTPUT
%token <token>    HALT
%token <token>    SKIPCOND
%token <token>    JUMP
%token <token>    CLEAR
%token <token>    ADDI
%token <token>    JUMPI
%token <token>    LOADI
%token <token>    STOREI
%token <token>    DEC
%token <token>    HEX
%token <token>    ORG
%token <token>    END
%token <token>    REGULAR_EXPRESSIONS
%token <token>    IDENTIFIER
%token <token>    HEXLIT
%token <token>    TOKEN_END
%%
program:
  statement_list
  {  F.tfs << endl << "#001 program -> statement_list";
     length=address-origin;
     F.tfs << endl << "length = " << hex << length;
     MemoryPrint(F.tfs);
     LT.Print(F.tfs);
     MemoryWrite(F.e);
  }
statement_list:
  statement
  {F.tfs << endl << "#002 statement_list -> statement";
  }
statement_list:
  statement_list statement
  {F.tfs << endl << "#003 statement_list -> statement_list statement";
  }
statement:
  directive
  {F.tfs << endl << "#004 statement -> directive";
  }
statement:
  labeled_item
  {F.tfs << endl << "#005 statement -> labeled_item";
  }
statement:
  item
  {F.tfs << endl << "#006 statement -> item";
  }
directive:
  ORG HEXLIT
  {F.tfs << endl << "#007 directive -> ORG HEXLIT(" << (*$2) << ")";
   F.tfs << endl;
   F.tfs << endl << (*$2) << "=" << hextoint(*$2) << " decimal";
   origin=hextoint(*$2);
   address=origin;
   F.tfs << endl << "origin=" << origin;
  }
labeled_item:
  label item
  {F.tfs << endl << "#008 labeled_item -> label item";
  }
```

```
  IDENTIFIER COMMA
  {F.tfs << endl << "#009 label -> identifier(" << (*$1) << ")" << "  ,";
   LT.Define(*$1,address,Memory);
  }
item:
  instruction
  {F.tfs << endl << "#010 item -> instruction ";
   F.tfs << endl << "address=" << address;
   address++;
  }
item:
  data_definition
  {F.tfs << endl << "#011 item -> data_definition";
   address++;
  }
instruction:
  JNS operand
  {F.tfs << endl << "#012 instruction -> JNS operand";
   unsigned short operation=op_jns|($2);
   F.tfs << endl << "instruction=" << setw(4) << hex << operation;
   F.tfs << dec;
   Memory[address]=operation;
  }
instruction:
  LOAD operand
  {F.tfs << endl << "#013 instruction -> LOAD operand";
   unsigned short operation=op_load|($2);
   F.tfs << endl << "instruction=" << setw(4) << hex << operation;
   F.tfs << dec;
   Memory[address]=operation;
  }
instruction:
  STORE operand
  {F.tfs << endl << "#014 instruction -> STORE operand";
   unsigned short operation=op_store|($2);
   F.tfs << endl << "instruction=" << setw(4) << hex << operation;
   F.tfs << dec;
   Memory[address]=operation;
  }
instruction:
  ADD operand
  {F.tfs << endl << "#015 instruction -> ADD operand";
   unsigned short operation=op_add|($2);
   F.tfs << endl << "instruction=" << setw(4) << hex << operation;
   F.tfs << dec;
   Memory[address]=operation;
  }
instruction:
  SUBT operand
  {F.tfs << endl << "#016 instruction -> SUBT operand";
   unsigned short operation=op_subt|($2);
   F.tfs << endl << "instruction=" << setw(4) << hex << operation;
   F.tfs << dec;
   Memory[address]=operation;
  }
instruction:
  INPUT
  {F.tfs << endl << "#017 instruction -> INPUT";
   unsigned short operation=op_input;
   F.tfs << endl << "instruction=" << setw(4) << hex << operation;
   Memory[address]=operation;
   F.tfs << dec;
  }
instruction:
  OUTPUT
```

<<<< USERNAME:tt FILENAME: maspar.y >>>>

```
    unsigned short operation=op_output;
    F.tfs << endl << "instruction=" << setw(4) << hex << operation;
    F.tfs << dec;
    Memory[address]=operation;
   }
instruction:
   HALT
   {F.tfs << endl << "#019 instruction -> HALT";
    unsigned short operation=op_halt;
    F.tfs << endl << "instruction=" << setw(4) << hex << operation;
    F.tfs << dec;
    Memory[address]=operation;
   }
instruction:
   SKIPCOND operand
   {F.tfs << endl << "#020 instruction -> SKIPCOND operand";
    unsigned short operation=op_skipcond|($2);
    F.tfs << endl << "instruction=" << setw(4) << hex << operation;
    F.tfs << dec;
    Memory[address]=operation;
   }
instruction:
   JUMP operand
   {F.tfs << endl << "#021 instruction -> JUMP operand";
    unsigned short operation=op_jump|($2);
    F.tfs << endl << "instruction=" << setw(4) << hex << operation;
    F.tfs << dec;
    Memory[address]=operation;
   }
instruction:
   CLEAR
   {F.tfs << endl << "#022 instruction -> CLEAR";
    unsigned short operation=op_clear;
    F.tfs << endl << "instruction=" << setw(4) << hex << operation;
    F.tfs << dec;
    Memory[address]=operation;
   }
instruction:
   ADDI operand
   {F.tfs << endl << "#023 instruction -> ADDI operand";
    unsigned short operation=op_addi|($2);
    F.tfs << endl << "instruction=" << setw(4) << hex << operation;
    F.tfs << dec;
    Memory[address]=operation;
   }
instruction:
   JUMPI operand
   {F.tfs << endl << "#024 instruction -> JUMPI operand";
    unsigned short operation=op_jumpi|($2);
    F.tfs << endl << "instruction=" << setw(4) << hex << operation;
    F.tfs << dec;
    Memory[address]=operation;
   }
instruction:
   LOADI operand
   {F.tfs << endl << "#025 instruction -> LOADI operand";
    unsigned short operation=op_loadi|($2);
    F.tfs << endl << "instruction=" << setw(4) << hex << operation;
    F.tfs << dec;
    Memory[address]=operation;
   }
instruction:
   STOREI operand
   {F.tfs << endl << "#026 instruction -> STOREI operand";
    unsigned short operation=op_storei|($2);
```

```
    F.tfs << dec;
    Memory[address]=operation;
   }
instruction:
   END
   {F.tfs << endl << "#027 instruction -> END";
   }
operand:
   HEXLIT
   {F.tfs << endl << "#028 operand -> HEXLIT(" << (*$1) << ")";
    $$=hextoint(*$1);
   }
operand:
   IDENTIFIER
   {F.tfs << endl << "#029 operand -> IDENTIFIER";
    $$=LT.Reference(*$1,address);
   }
data_definition:
   HEX HEXLIT
   {F.tfs << endl << "#030 data_definition -> HEX HEXLIT(" << (*$2) << ")";
    F.tfs << endl << (*$2) << "=" << hextoint(*$2) << " decimal";
    Memory[address]=hextoint(*$2);
   }
data_definition:
   DEC HEXLIT
   {F.tfs << endl << "#031 data_definition -> DEC HEXLIT(" << (*$2) << ")";
    F.tfs << endl << (*$2) << "=" << dectoint(*$2) << " decimal";
    Memory[address]=dectoint(*$2);
   }
%%
//-----------------------------------------------------------------------
//User function section
//-----------------------------------------------------------------------
void yyerror(const char* m)
{    cout << endl
          << "line(" << Line << ") col(" << Col << ") " << m;
     cout << endl;
}
void MemoryTitle(ostream& o)
{    o << endl;
     o << "adr";
     for (int a=0;a<16;a++) {
          o << " ";
          o << "ad";
          o << setfill('0') << hex << setw(2) << a;
     }
     o << setfill(' ') << dec;
}
void MemoryPrint(ostream& o)
{    MemoryTitle(o);
     for (int a=origin;a<origin+length;a++) {
          if (a%16==0) {
               o << endl;
               o << setw(3) << setfill('0') << hex << a;
          }
          o << " ";
          o << setw(4) << setfill('0') << hex << Memory[a];
    }
    o << endl;
    o << setfill(' ') << dec;
}
void MemoryWrite(FILE* f)
{
     fwrite(&origin,sizeof(origin),1,f);
     F.tfs << endl << "origin = " << hex << origin << endl;
```

```
    F.tfs << endl << "length = " << hex << address << endl;
    for (int a=origin;a<origin+length;a++) {
        fwrite(&Memory[a],sizeof(short int),1,f);
        F.tfs << endl;
        F.tfs << setw(4) << hex <<  a;
        F.tfs << "  ";
        F.tfs << setw(4) << hex << Memory[a];
    }
}
```