Práctica 6
Bosquejo de un traductor de instrucciones de ensamblador a lenguaje máquina

Introducción

La traducción de ensamblador a lenguaje máquina parece ser algo sencillo por la correspondencia 1 a 1 entre estos dos lenguajes de alto nivel. Sin embargo, dependiendo de la arquitectura del procesador, esto puede complicarse un poco. Tal es el caso de la arquitectura de Intel de 32 bits, IA32, que por mantener compatibilidad hacia atrás se ha vuelto una arquitectura muy rebuscada.
En esta práctica capturaremos un primer bosquejo de un traductor de instrucciones de ensamblador a lenguaje máquina para esta arquitectura. El formato de instrucciones para los procesadores de 16 bits de esta familia es:

CodOperación[d,v,s,w] + mod [reg] r/m + [byte bajo dato] + [byte alto dato, si w=1]
(1 byte) (1 byte) (1 byte) (1 byte)

Existen instrucciones que no tienen operandos, otras con un operando y otras con 2 operandos. El tamaño de la instrucción es variable y dado que el manejo de las palabras en memoria es “liitle endian”, en el formato de instrucción las palabras quedan con sus bytes invertidos.

Objetivos:

Capturar, compilar y probar los analizadores léxicos y sintácticos de este programa traductor de instrucciones de ensamblador a lenguaje máquina.

Actividades:

1.- Captura el analizador sintáctico siguiente:

```
/* Archivo: asm16.y
 * Analizador lexico para el ensamblador de 16 bits.
*/
%{
#include<stdio.h>
#include<string.h>
%}

%token  COMA ETIQ REG8 REG16 AC8 AC16 INM8 INM16 RSEG COMM
%token  CBW CWD
%token  ADD MUL MOV
%token  ERROR

%%

Prog:
     ListaInst
     | Prog ListaInst
     | "END"         { return 0; }
     ;

ListaInst:
     CBW  { printf("Leng. maquina = 0x98\tTam=1\n"); }
     | CWD { printf("Leng. maquina = 0x99\tTam=1\n"); }
     | MUL REG8 { printf("Leng. maquina = %x\tTam=2\n",0xf6e0|$2); }
     | MUL REG16 { printf("Leng. maquina = %x\tTam=2\n",0xf7e0|$2); }
     | ADD REG8 COMA REG8 { $$ = 0x02C0|($2 << 3)|$4; printf("Leng.
maquina = %x\tTam=2\n",$$);}
     | ADD REG16 COMA REG16     { $$ = 0x03C0|($2 << 3)|$4;printf("Leng.
maquina = %x\tTam=2\n",$$); }
     | MOV REG8 COMA REG8  { $$ = 0x8AC0|($2 << 3)|$4;printf("Leng.
```

```
maquina = %x, Tam = 2\n",$$);}
      | MOV REG16 COMA REG16     { $$= 0x8BC0|($2<<3)|$4;printf("Leng.
maquina = %x, Tam = 2\n",$$); }
      | COMM          { ; /* elimina comentarios */ }
      | ERROR         { printf("Error desconocido\n"); }
      ;
%%

main()
{
      printf("Traductor de instrucciones de ensamblador.\n");
      yyparse();
}

yyerror(char *msje)
{
      printf("Error: %s\n",msje);
}
```

1.1.- Desde una ventana de DOS compila este programa yacc con el comando

```
bison -d asm16.y
```

Este comando debe generar los archivos `asm16.tab.y` y `asm16.tab.h.` Compruébalo con el comando `dir`

Archivo	Tamaño
asm16.tab.c	_____
asm16.tab.h	_____

2.- Captura el analizador léxico siguiente:

```
/* Archivo: asm16.l
 * Analizador lexico para ensamblador de 16 bits
*/
%{
#include <stdio.h>
#include<stdlib.h>
#include "asm16.tab.h"
#include "asm16.h"

%}
%%
[ \t]+          {/* se ignoran los espacios en blanco */}
[0-9]+          {   yylval =atoi(yytext);
                if (yylval < 256) return(INM8);
                else if (yylval < 65536) return(INM16);
                else return(ERROR);
            }
";""\n"  { ; }
";"[ \t][a-zA-Z0-9]*"\n"  { ; }
","         { return(COMA); }
[a-zA-Z]+":"    { return (ETIQ); }
[aA][lL]        { yylval= 0;  return(AC8);}
[aA][hH]        { yylval = 4; return(REG8); }
[aA][xX]        { yylval = 0; return(AC16); }
[bB][lL]        { yylval = 3; return(REG8); }
[bB][hH]   { yylval = 7; return(REG8); }
[bB][xX]        { yylval = 3; return(REG16); }
[cC][lL]        { yylval = 1; return(REG16); }
[cC][hH]        { yylval = 5; return(REG8); }
```

```
[cC][xX]   { yylval = 1; return(REG16); }
[dD][lL]   { yylval = 2; return(REG8); }
[dD][hH]   { yylval = 6; return(REG8); }
[dD][xX]   { yylval = 2; return(REG16); }
[sS][pP]   { yylval = 4; return(REG16); }
[bB][pP]   { yylval = 5; return(REG16); }
[sS][iI]   { yylval = 6; return(REG16); }
[dD][iI]   { yylval = 6; return(REG16); }

[cC][bB][wW]    { return(CBW); }
[cC][wW][dD]    { return(CWD); }
[mM][uU][lL]    { return(MUL); }
[aA][dD][dD]    { return(ADD); }
[mM][oO][vV]    { return(MOV); }
%%

int yywrap(void)
{
     return 1;
}
```

2.1.- Desde una ventana de DOS compila este programa lex con el comando:

```
flex -oasm16.c asm16.l
```

Esto debe generar el archivo `asm16.c`. Compruébalo con el comando `dir`:

Archivo Tamaño

`asm16.c` _____

3.- Ahora genera el ejecutable asm16.exe utilizando el compilador de C que eligas.

Comando a usar: _____

Ejecútalo y comprueba que se haya generado el archivo asm16.exe

Archivo Tamaño

`asm16.exe` _____

4.- Para probar el programa `asm16.exe` captura el siguiente archivo con instrucciones de prueba y llámalo `prueba.asm`:

```
; programa de prueba
cbw
cwd
mul bh
mul BX
add bh,cl
add cx,bx
mov cl,dh
mov bx,CX
END
```

4.1.- Para probar la traducción de instrucciones desde una ventana de DOS teclea el siguiente comando:

```
asm16 < prueba.asm
```

y reporta los resultados que obtengas:

_____

_____

_____

_____

5.- Ahora modifica los analizadores léxico y sintáctico para agregar las instrucciones:

clc : CodOp: 1111 1000

div reg8 : CodOp: 1111 0110 1111 0reg

5.1.- Modificaciones a `asm16.y` (analizador sintáctico):

_____

_____

_____

5.2.- Modificaciones a `asm16.l` (analizador léxico):

_____

_____

_____

5.3.- Recompila ambos programas (`asm16.y` y `asm16.l`) y vuelve a generar el ejecutable asm16.exe y reporta los datos de los archivos obtenidos:

Archivo	Tamaño
`asm16.tab.c`	_____
`asm16.tab.h`	_____
`asm16.c`	_____
`asm16.exe`	_____

5.4.- Ahora modifica el archivo `prueba.asm` para incluir una instrucción de prueba para cada una de las instrucciones agregadas, y desde una ventana de DOS teclea el siguiente comando:

```
asm16 < prueba.asm
```

y reporta los resultados que obtengas:

_____

_____

_____

_____

6.- Comentarios y conclusiones

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____