6. [1.0 puntos] Dada la siguiente gramática que define un lenguaje ensamblador de un procesador RISC:

program	→	head block
head	→	**'CODE'** addr addr
block	→	instr block
	|	instr
instr	→	**'LOAD'** reg addr
	|	**'STORE'** addr reg
	|	**'STORE'** addr const
	|	**'MOVE'** reg reg
	|	**'ADD'** reg reg reg
	|	**'SUB'** reg reg reg
	|	**'CMP'** reg reg
	|	**'JMP'** const
	|	**'HALT'**
reg	→	**'R1'** | **'R2'** | **'R3'** | **'R4'**
addr	→	'#' const
const	→	const num
	|	num
num	→	"0" | "1" | ··· | "9"

Obtener la tabla de tokens con el máximo nivel de abstracción suponiendo que se va a realizar:

(a) [0.8 puntos] Traducción.

Solución:

La tabla de tokens con el máximo nivel de abstracción cuando se realiza el proceso completo de traducción sería la siguiente:

Nº	Token	Patrón	Atributos
1	CODE	"CODE"	
2	LOAD	"LOAD"	
3	STORE	"STORE"	
4	MOVECMP	"MOVE"|"CMP"	0:move, 1:cmp
5	ADDSUB	"ADD"|"SUB"	0:add, 1:sub
6	JMP	"JMP"	
7	HALT	"HALT"	
8	REG	"R1"|"R2"|"R3"|"R4"	0:R1, 1:R2, 2:R3, 3:R4
9	ADDR	'#'[0-9]+	
10	CONST	[0-9]+	

(b) [0.2 puntos] Análisis sintáctico.

Solución:

Cuando únicamente se va a realizar análisis sintáctico, no hay que distinguir palabras según su valor semántico elemental ni considerar secuencias de palabras atendiendo a determinados valores semánticos estructurados.

Se observa que la gramática original se rige bajo las condiciones de una **gramática regular**, por lo tanto, sería posible expresar la gramática mediante una única expresión regular. Se empleará la sintaxis de LEX para especificar el patrón:

```
%option noyywrap

sep     ([ \t\n])*
cons    [0-9]+
addr    "#"{cons}
reg     "R"[1234]
head    "HEAD"{sep}{addr}{sep}{addr}{sep}
load    "LOAD"{sep}{reg}{sep}{reg}
store   "STORE"{sep}{addr}{sep}({reg}|{cons})
movcmp  ("MOVE"|"CMP"){sep}{reg}{sep}{reg}
addsub  ("ADD"|"SUB"){sep}{reg}{sep}{reg}{sep}{reg}
jmp     "JMP"{sep}{cons}
halt    "HALT"
instr   (({load}|{store}|{movcmp}|{addsub}|{jmp}|{halt}){sep})*

%%
[ \t\n]+                        ;
{head}{sep}({instr})*           { ECHO; printf("\n *** Secuencia Correcta ***\n");  }
.                               { printf("[Linea %d] Error en %s\n", yylineno, yytext); }
%%
```

7. 0.25 puntos Especificar la gramática abstracta del problema 6(a) usando la sintaxis de YACC y realizar las modificaciones adicionales para que considere recuperación de errores en la entrada ante las siguientes situaciones:

Solución:

En primer lugar vamos a especificar la gramática abstracta usando la sintaxis de YACC:

```
%token CODE LOAD STORE MOVECMP ADDSUB
%token JMP HALT REG ADDR CONST
%start program
%%
program : head block ;
head    : HEAD ADDR ADDR ;
block   : instr block
        | instr ;
instr   : LOAD REG ADDR
        | STORE ADDR REG
        | STORE ADDR CONST
        | MOVECMP REG REG
        | ADDSUB REG REG REG
        | JMP CONST
        | HALT ;
%%
```

(a) 0.10 puntos Cuando acontezca un error en una instrucción (`instr`).

Solución:

Cuando aparezca un error por no poder construir sintácticamente las palabras que deben aparecer en una instrucción, debemos añadir la siguiente producción:

```
%token CODE LOAD STORE MOVECMP ADDSUB
%token JMP HALT REG ADDR CONST
%start program
%%
program : head block ;
head    : HEAD ADDR ADDR ;
block   : instr block
        | instr ;
instr   : LOAD REG ADDR
        | STORE ADDR REG
        | STORE ADDR CONST
        | MOVECMP REG REG
        | ADDSUB REG REG REG
        | JMP CONST
        | HALT
        | error ;    /* Producción de error */
%%
```

(b) 0.15 puntos Cuando acontezca un error en la referencia a los registros que aparecen como parámetro de las instrucciones.

Solución:

Cuando aparezca un error debido a que no aparece la especificación de un registro (token `REG`) nos diría que ha aparecido cualquier otro token menos el del propio registro. Para esta situación, dado que `REG` es un token, debemos considerar un nuevo símbolo no terminal que nos lleve a la formación de `REG` o, si no aparece, se pueda recuperar el análisis indicando la producción de error ante ese nuevo no terminal que se crea. Por último, cualquier aparición del token `REG` deberá sustituirse por el nuevo no terminal `reg`:

```
%token CODE LOAD STORE MOVECMP ADDSUB
%token JMP HALT REG ADDR CONST
%start program
%%
program : head block ;
head    : HEAD ADDR ADDR ;
block   : instr block
        | instr ;
instr   : LOAD reg ADDR     /* Se sustituye REG por el no terminal ''reg'' */
        | STORE ADDR reg
        | STORE ADDR CONST
        | MOVECMP reg reg
        | ADDSUB reg reg reg
        | JMP CONST
        | HALT
        | error ;
reg     : REG         /* Un registro es REG */
        | error ;     /* en otro caso, error */
%%
```

8. 0.5 puntos Dada la gramática abstracta del problema 6(a) usando la sintaxis de YACC, realizar los cambios que se estimen oportunos, construir la tabla de análisis LL(1) y verificar si dicha gramática cumple los requisitos para ser analizada empleando esta estrategia de análisis.

Solución:

Para realizar un análisis descendente mediante la estrategia de análisis LL(1), debemos considerar dos aspectos concretos en la gramática abstracta de partida:

- Recursividad a la izquierda.
- Factorización.

El primero de ellos no acontece en la gramática pero el problema de la factorización sí que aparece en dos lugares. Una es cuando se especifica el no terminal `block`:

```
....
block   : instr block
        | instr ;
....
```

La solución sería la siguiente:

```
....
block   : instr block2
block2  : block
        |  ;
....
```

La otra factorización acontece en el no terminal `instr` en cuanto a sus opciones `STORE` y en el siguiente token `ADDR`.

```
....
instr   : ....
        | STORE ADDR REG
        | STORE ADDR CONST
        ....
....
```

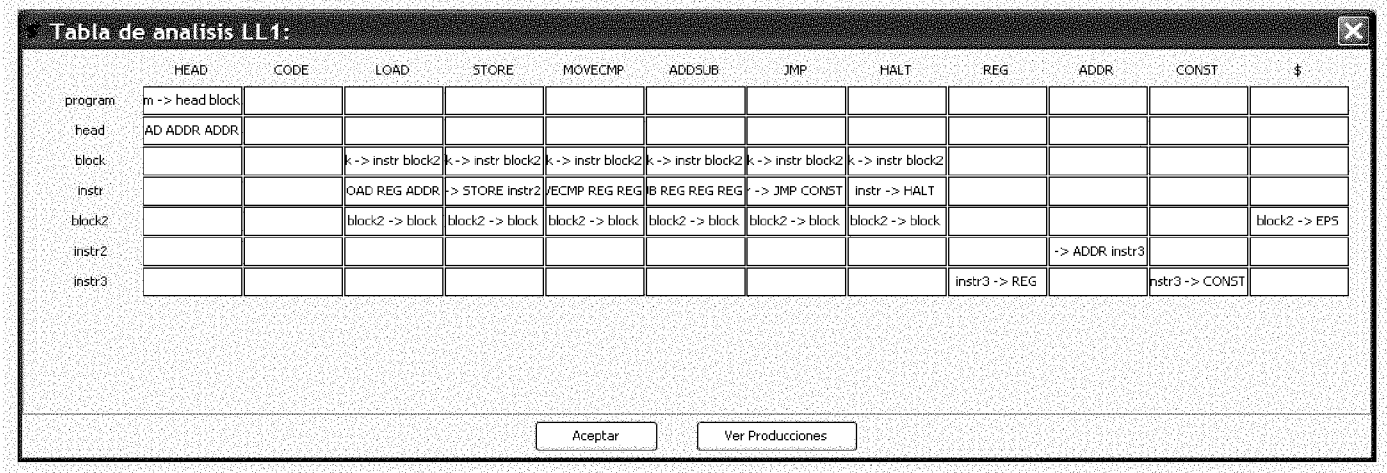
La solución sería la siguiente:

```
....
instr   : .....
        | STORE instr2
        | .....
instr2  : ADDR REG
        | ADDR CONST ;
....
```

Tendríamos que factorizar de nuevo en `instr2` dado que comienzan por el mismo terminal `ADDR`. La solución final sería la siguiente:

```
....
instr   : .....
        | STORE instr2
        | .....
instr2  : ADDR instr3 ;
instr3  : REG
        | CONST ;
....
```

Una vez calculados los iniciales y seguidores de aquellas producciones en las que aparezcan varias alternativas, la tabla de análisis LL(1) quedaría así (obtenida mediante Sefalas):



Por lo tanto, ante la ausencia de conflictos, esta gramática resultante es LL(1).