UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Facultad de Ciencias

Integrantes:
Adrián Aguilera Moreno
Sebastián Alejandro Gutierrez Medina

Compiladores

Tarea 06

Varios lenguajes de programación, por ejemplo C, tienen definido el enunciado `switch`:

```
switch E
   begin
      case V1 : S1
      case V2 : S2
      ...
      case Vn1 : Sn1
   default: Sn
end
```

Describe una forma de traducir este enunciado a código de tres direcciones (puedes usar saltos y condicionales). Explica y justifica que el código de tres direcciones propuesto respeta el mismo comportamiento que el `switch`.

Considera el siguiente código:

```
int mul(int x, int y) {
if (x) return y - (0 - mul(x + 1, y));
   else return 0; }
```

Proporciona una traducción a RTL suponiendo que ya se han seleccionado las instrucciones. Deberás mostrar las traducciones intermedias de subexpresiones del código.

Utilizando la tabla de traducción entre una representación intermedia lineal y las instrucciones de arquitectura MIPS, genera el código máquina para la siguiente secuencia:

$t := r_s + k$, $r_t := M[t^{last}]$		`lw`	r_t, $k(r_s)$
$r_t := M[r_s]$		`lw`	r_t, $0(r_s)$
$r_t := M[k]$		`lw`	r_t, k(R0)
$t := r_s + k$, $M[t^{last}] := r_t$		`sw`	r_t, $k(r_s)$
$M[r_s] := r_t$		`sw`	r_t, $0(r_s)$
$M[k] := r_t$		`sw`	r_t, k(R0)
$r_d := r_s + r_t$		`add`	r_d, r_s, r_t
$r_d := r_t$		`add`	r_d, R0, r_t
$r_d := r_s + k$		`addi`	r_d, r_s, k
$r_d := k$		`addi`	r_d, R0, k
`GOTO` *label*		`j`	*label*
`IF` $r_s = r_t$ `THEN` $label_t$ `ELSE` $label_f$, `LABEL` $label_f$		`beq`	r_s, r_t, $label_t$
	$label_f$:		
`IF` $r_s = r_t$ `THEN` $label_t$ `ELSE` $label_f$, `LABEL` $label_t$		`bne`	r_s, r_t, $label_f$
	$label_t$:		
`IF` $r_s = r_t$ `THEN` $label_t$ `ELSE` $label_f$		`beq`	r_s, r_t, $label_t$
		`j`	$label_f$
`IF` $r_s < r_t$ `THEN` $label_t$ `ELSE` $label_f$, `LABEL` $label_f$		`slt`	r_d, r_s, r_t
		`bne`	r_d, R0, $label_t$
	$label_f$:		
`IF` $r_s < r_t$ `THEN` $label_t$ `ELSE` $label_f$, `LABEL` $label_t$		`slt`	r_d, r_s, r_t
		`beq`	r_d, R0, $label_f$
	$label_t$:		
`IF` $r_s < r_t$ `THEN` $label_t$ `ELSE` $label_f$		`slt`	r_d, r_s, r_t
		`bne`	r_d, R0, $label_t$
		`j`	$label_f$
`LABEL` *label*	*label*:		

Hasta 1.5pts extra. Determina el código de tres direcciones de la siguiente expresión,

$$(a\ SUB\ b)\ MOD\ ((MINUS\ c)\ SUB\ d)$$

usando las reglas semánticas siguientes.
Pueden omitir la explicación de la creación del árbol de sintaxis abstracta, pero hay que explicar los pasos del análisis semántico.

Synthesized Attributes

E.code:	Code sequence evaluating *E*
E.sym:	Symbol representing value of *E*
addop.*op*:	addition operator: **ADD** or **SUB**
mulop.*op*:	multiplication operator: **MUL**, **DIV**, or **MOD**

Grammar Rules	Semantic Rules
$E \rightarrow E^1$ **addop** E^2	*E.sym* := newtemp(); *AddInst* := **new** Inst(**addop**.*op*, *E.sym*, E^1.*sym*, E^2.*sym*); *E.code* := E^1.*code* + E^2.*code* + *AddInst*;
$E \rightarrow E^1$ **mulop** E^2	*E.sym* := newtemp(); *MulInst* := **new** Inst(**mulop**.*op*, *E.sym*, E^1.*sym*, E^2.*sym*); *E.code* := E^1.*code* + E^2.*code* + *MulInst*;
$E \rightarrow$ *UnaryOp* E^1	*E.sym* := newtemp(); *UnaryInst* := **new** Inst(*UnaryOp.op*, *E.sym*, E^1.*sym*); *E.code* := E^1.*code* + *UnaryInst*;

Synthesized Attributes

UnaryOp.op:	Unary operator: **PLUS**, **MINUS**, **NOT**
id.*name*:	Identifier name
num.*sym*:	Literal symbol holding number value

Grammar Rules	Semantic Rules
$E \rightarrow$ **(** E^1 **)**	*E.sym* := E^1.*sym*; *E.code* := E^1.*code*;
$E \rightarrow$ **id**	*E.sym* := *idTable*.lookup(**id**.*name*); *E.code.first* := *E.code.last* = 0;
$E \rightarrow$ **num**	*E.sym* := **num**.*sym*; *E.code.first* := *E.code.last* = 0;
UnaryOp $\rightarrow$ **addop**	**if addop**.*op* = **ADD then** *UnaryOp.op* := **PLUS** **else** *UnaryOp.op* := **MINUS**
UnaryOp $\rightarrow$ **not**	*UnaryOp.op* := **NOT**