TD2 de compilation
Assembleur

Exercice 1 (mJS-code).
Écrire un code assembleur pour la mini-JS-machine traduisants les programmes suivants ; mis à part pour le premier, exécutez le code à la main :

— d'abord une expression arithmétique simple : [1]
```
25.3 + 7 * -(5E-3 === .2 ? (25 * -.4) + 2 : 42 / 2.45 )
```
— plus comliqué :
```
x = 12;
Si (12 + 3 <= x * 2) {
  y = 30;
  TantQue (y > x - 3) y = y-x ;
} Sinon y = x;
```
— avec des changements de types :
```
x = 12; y = 2;
TantQue (y > 0) {
  x = x + y + "log" ;
  y = y + 1;
}
Ecrire(x);
```
— avec une fonction :
```
Var x = 12;
Var y = 2;
Ecrire (foo(x) + foo("test" + y));
Fonction foo (x) {
  y = y - 1;
  Si (0 > y) Retourner (x + 3);
  Sinon Retourner foo(x+y);
}
```

Instruction	sémentique	pile avant	pile après
AddiRe	Push(Pop $+_f$ Pop);	#r:#r:pile	#r:pile
SubsRe	Push(Pop $-_f$ Pop);	#r:#r:pile	#r:pile
MultRe	Push(Pop $*_f$ Pop);	#r:#r:pile	#r:pile
DiviRe	Push(Pop$/_f$Pop);	#r:#r:pile	#r:pile
NegaRe	Push(Pop $*_f$ (-1));	#r:pile	#r:pile
ReToBe	DoubleToBool(Pop);	#r:pile	#b:pile
Equal	Push(Pop $==$ Pop);	#v:#v:pile	#b:pile
LowEqR	Push(Pop $\leq_f$ Pop);	#r:#r:pile	#b:pile
GreStR	Push(Pop $>$ Pop);	#r:#r:pile	#b:pile

1. Ici on sait qu'il n'y a que des flotants, on se place dans le fragment 1, sans strings.

Instruction	sémentique	pile avant	pile après
Jump offset	PC := PC + off + 1;	pile	pile

ConJmp offset	if Pop
	then PC := PC + 1;
	else PC := PC + off + 1;
		#b:pile	pile

Case	PC := PC + Floor(Pop) + 1;	#r:pile	pile
CstRe x	Push(x);	pile	#f:pile
CstBo x	Push(x);	pile	#b:pile
Copy	Push(Pull);	X:pile	X:X:pile

Swap	#vx := Pop;
	#vy := Pop;
	Push(x);
	Push(y);
		X:Y:pile	Y:X:pile

Drop	Pop;	X:pile	pile
TypeOf	Pull une valeur, rend 0 sur un booleen, 1 sur un flottant, 2 sur une string, 3 sur undefined, 4 sur une cloture, 5 sur un objet,	X:pile	#r:X:pile
SetVar n	Set(n, Pull);	#v:pile	pile
GetVar n	Push(Get(n))	pile	#v:pile
Print	affiche Pop sur le terminal	#v:pile	pile

SetArg	#v v = Pop
	#l clot = Pull
	#n n = clot.args.Pop
	clot.cont.Insert(n, v)
	PC := PC + 1
		#v:#c:pile	#c:pile

Call	#l clot = Pop Push(NewContinuation{cont = CC, methode = PC, err = 0}) CC := clot.cont PC := clot.methode	#l:pile	#t:pile
Return	#v res = Pop; do {#t continue = Pop; } while (continue.err) CC := continue.cont PC := continue.methode Push(res)	X:autre@#t:pile	X:pile
DclVar n	Insert(n, undefind)	pile	pile
Lambda pos	Push(NewCloture{cont = CopyCont, methode = pos})	pile	#l:pile
DclArg n	Pull.args.Push(n)	#l:pile	#l:pile
Concat	Push(Pop +$_s$ Pop);	#s:#s:pile	#s:pile
CstStr x	Push(x);	pile	#s:pile
StToRe	StringToDouble(Pop);	#s:pile	#r:pile
ReToSt	DoubleToString(Pop);	#r:pile	#s:pile