Esterel de A à Z

4. Traduction d'Esterel en circuits

Gérard Berry

Collège de France
Chaire Algorithmes, machines et langages

Cours du 7 mars 2018

suivi du séminaire d'Albert Benveniste et Thierry Gautier

gerard.berry@college-de-france.fr
http://www-sop.inria.fr/members/Gerard.Berry

COLLÈGE
DE FRANCE
1530

Objectifs de ce cours (et du suivant)

- Expliquer la traduction d'Esterel en circuits Booléens
  - qui a résolu tous les problèmes initiaux du langage
  - qui a été industrialisée en grand, et appliquée à des logiciels et circuits complexes, commerciaux ou de recherche
  - qui permet aussi de traduire Esterel en Lustre
  - mais qui n'est pas la seule façon de compiler le langage en codes C ou autres (voir cours 6 pour bien plus efficace)
- Et rendre vraiment propre et formelle une opération fondamentale sur les circuits, le *clock-gating*
  - critique pour l'efficacité énergétique et le multi-horloges

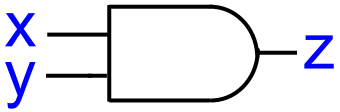
Merci a Jean Vuillemin pour avoir démarré tout ça,
Revoir les cours du 9 avril 2013 (circuits 2-adiques)

Plan du cours

1. Rappels sur les circuits synchrones

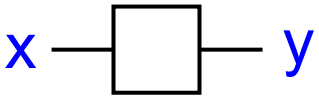
Portes combinatoires et registres

z = x or y



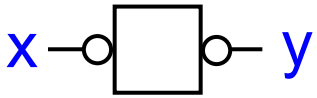
z = x and y

z = (not x) and y



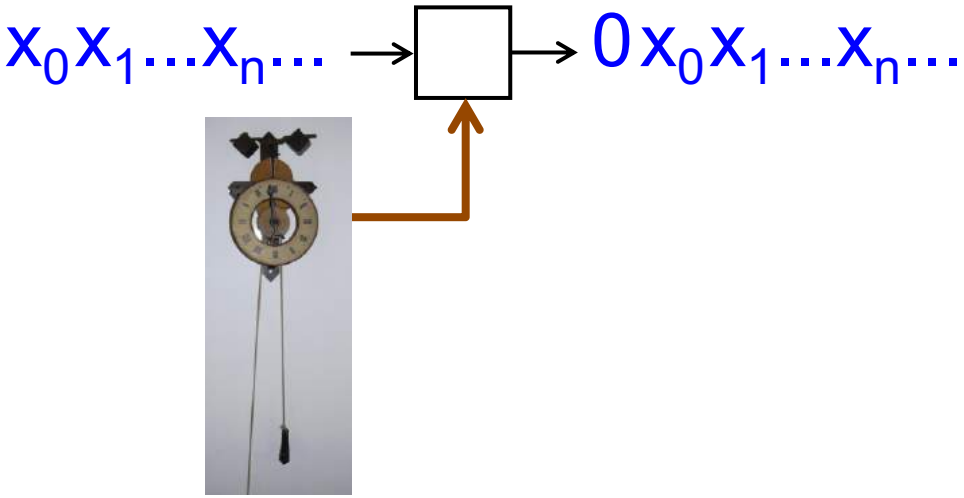
reg

y = 0→pre(x)

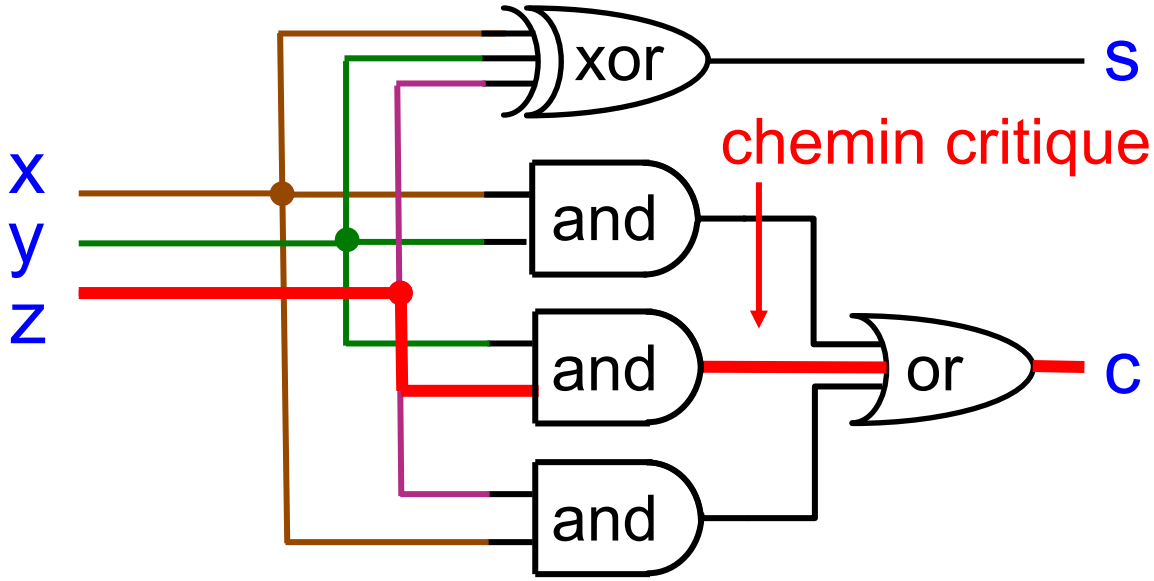


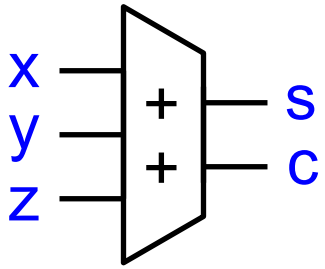
reg1

y = 1→pre(x)



Circuits combinatoires acycliques





Full Adder : s = x xor y xor z

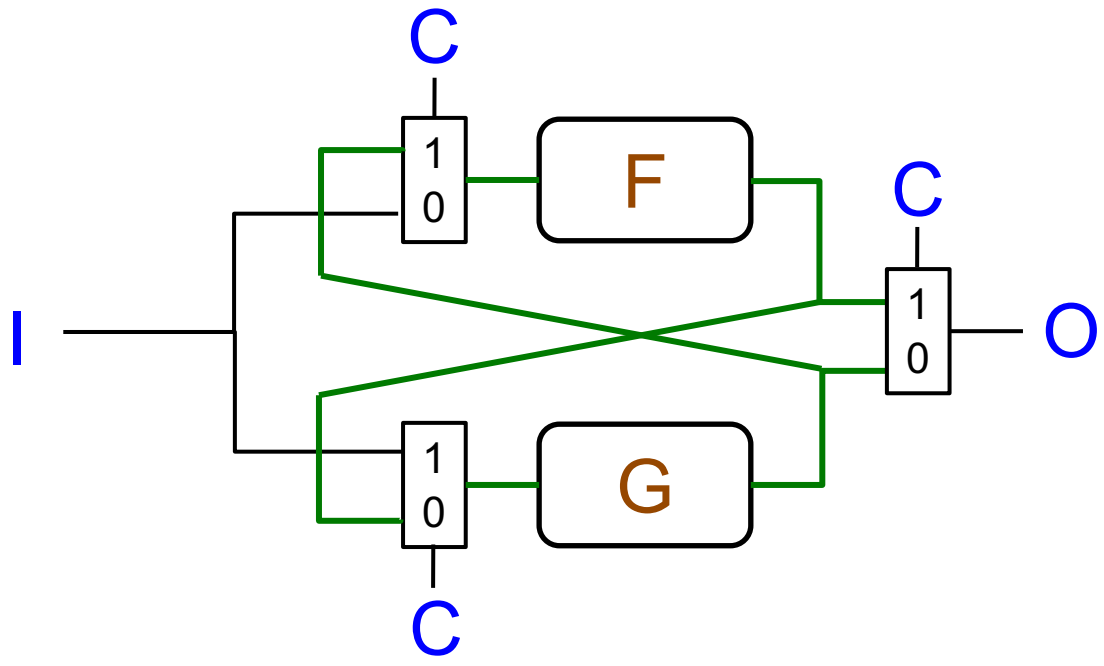
c = (x and y) or (y and z) or (z and x)

Si on garde les entrées stables à 0 ou 1,
alors, en temps fini, les sorties se stabilisent à 0 ou 1,
sur les seules valeurs vérifiant les équations

Circuits combinatoires cycliques

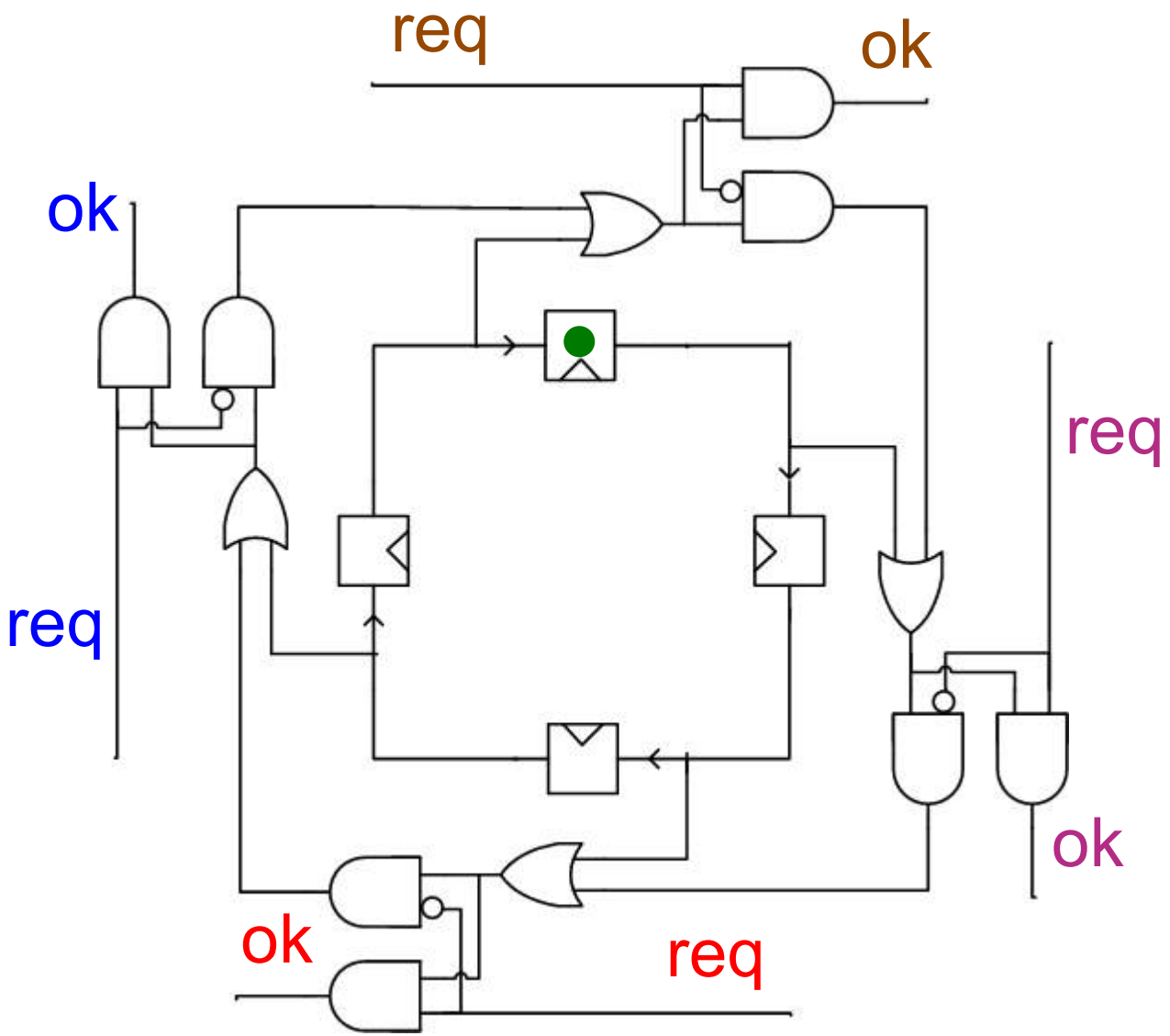
(cf. cours du 26/03/2014)

O = if C then F(G(I)) else G(F(I))



Electriquement sain ssi les sorties
se stabilisent en temps borné
⇔ ssi il se comporte comme un circuit acyclique

Circuits séquentiels à combinatoire cyclique



cycle cassé par le jeton tournant (28/03/2014)

Théorème fondamental (Mendler-Berry-Shiple)

Théorème : Un circuit combinatoire cyclique C est électriquement sain pour des entrées I si et seulement si ses sorties O peuvent être calculées à partir des équations de C en utilisant seulement la logique Booléenne constructive (sans tiers exclu)

Corollaire : Un circuit séquentiel à combinatoire cyclique C est électriquement sain si et seulement si ses états accessibles rendent sa partie combinatoire saine

Voir la preuve dans le cours du 26 mars 2014

Plan du cours

1. Rappels sur les circuits synchrones
2. La sémantique par termes marqués (états)

`nothing`	*ne fait rien et termine instantanément*
`emit` S	*émet le signal* S
`pause`	*arrête l'exécution,* *puis redémarre au prochain instant*
p `;` q	*exécute* p *puis* q *si et quand* p *termine*
`loop` p `end`	*répète* p *indéfiniment*
p `||` q	*exécute* p *et* q *en parallèle synchrone*
`trap` T `in` p `end`	*déclare et gère la trappe* T *dans* p
`exit` T^k	*lève la trappe* T *d'index* k
`if` S `then` p `else` q `end`	*exécute* p *si* S *est présent,* q *sinon*
`suspend` S `when` p	*lance* p *puis le suspend si* S *est présent* *termine si* p *termine* *et premier instant ou* S *absent*
`signal` S `in` p `end`	*déclare le signal* S *local dans* p

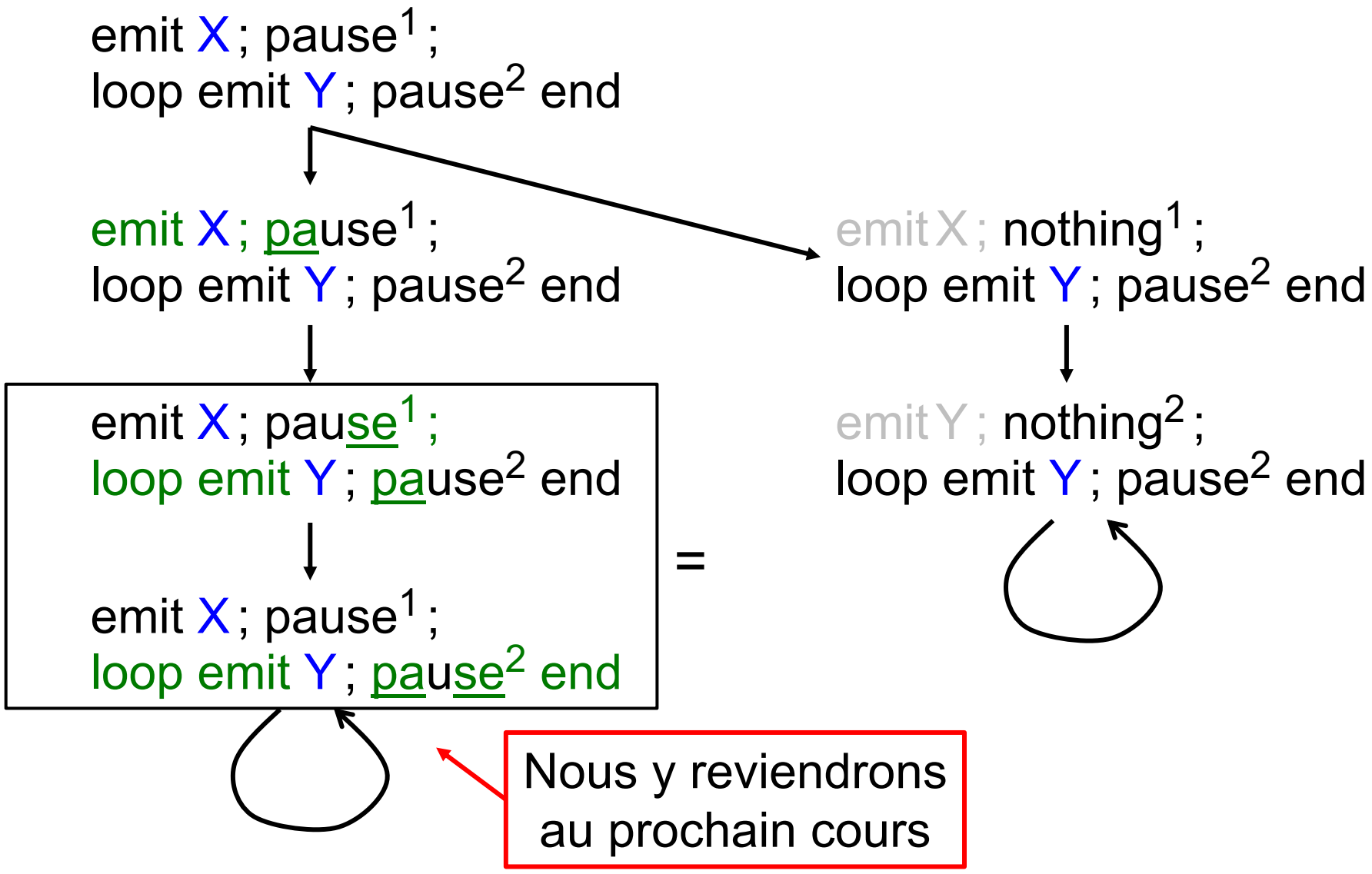
Deux étapes

1. Ce cours : traduction des circuits sans boucles (loop)
2. Cours du 14/03/2018 :
   – traitement des boucles et de la réincarnation des signaux et instructions
   – formalisation du clock gating par la suspension faible

Un traitement naïf des boucles donnerait des cycles non constructifs, donc électriquement instables

Prendre au sérieux la coloration des termes

(cours 3 du 14/02/2018)



Simplifier la coloration des termes

emit X ; pause^1 ;
loop emit Y ; pause^2 end

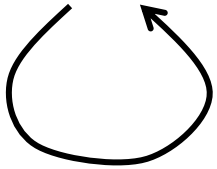
état { }

emit X ; pause^1 ;
loop emit Y ; pause^2 end

état {1}

emit X ; pause^1 ;
loop emit Y ; pause^2 end

état {2}



Georges Gonthier

Parallélisme : plusieurs pause^n possibles dans un état

Simplifier la coloration des termes

emit X ; pause^1 ;
loop emit Y ; pause^2 end

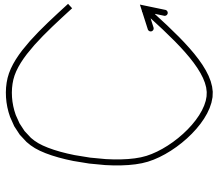
p

emit X ; pause^1 ;
loop emit Y ; pause^2 end

$\hat{p}^{\{1\}}$

emit X ; pause^1 ;
loop emit Y ; pause^2 end

$\hat{p}^{\{2\}}$



p : terme standard
$\hat{p}$: terme contenant au moins une pause activée
$\overline{p}$: p *ou* $\hat{p}$

nothing	
pause	pause
emit S	
exit T^k	
$p;q$	$\hat{p};q$ $p;\hat{q}$ ~~$\hat{p};\hat{q}$~~
$p \mid\mid q$	$\hat{p} \mid\mid q$ $p \mid\mid \hat{q}$ $\hat{p} \mid\mid \hat{q}$
trap T in p end	trap T in $\hat{p}$ end
if S then p else q end	if S then $\hat{p}$ else q end if S then p else $\hat{q}$ end ~~if S then $\hat{p}$ else $\hat{q}$ end~~
abort p when S	abort $\hat{p}$ when S
suspend p when S	suspend $\hat{p}$ when S
signal S in p end	signal S in $\hat{p}$ end

Deux sortes de règles

go : démarrage d'une instruction non activée p
res (*resume*) : reprise d'une instruction activée $\hat{p}$

$$\texttt{nothing} \xrightarrow[I]{\emptyset, 0} \texttt{nothing} \qquad (go\text{-}nothing)$$

$$\texttt{pause} \xrightarrow[I]{\emptyset, 1} \underline{\texttt{pause}} \qquad (go\text{-}pause)$$

$$\underline{\texttt{pause}} \xrightarrow[I]{\emptyset, 0} \texttt{pause} \qquad (res\text{-}pause)$$

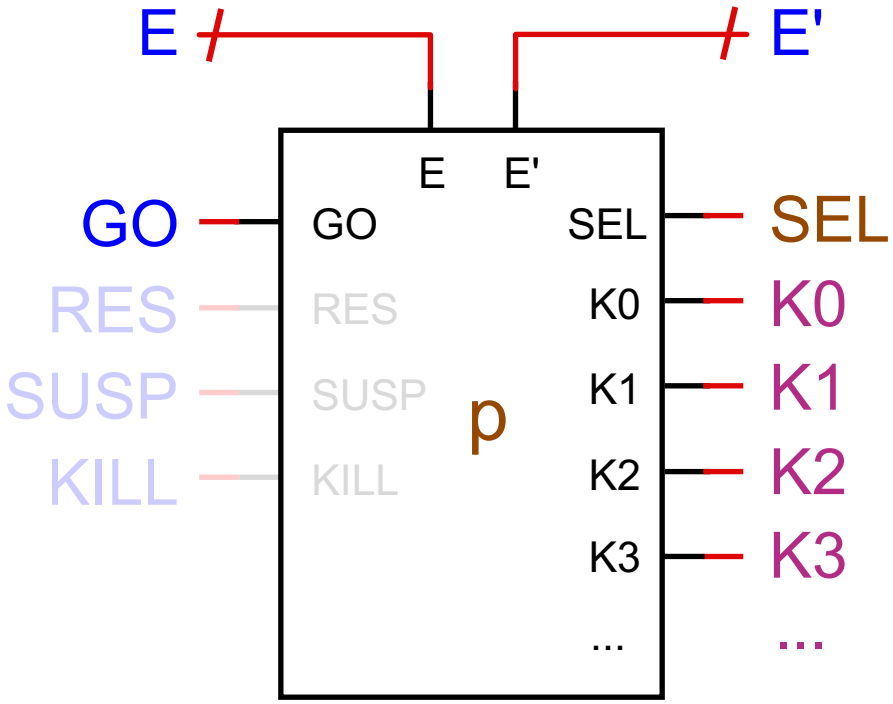
$$\texttt{exit}\ T^k \xrightarrow[I]{\emptyset, k} \texttt{nothing} \qquad (go\text{-}exit)$$

$$\texttt{emit}\ S \xrightarrow[I]{\{S\}, 0} \texttt{nothing} \qquad (go\text{-}emit)$$

Plan du cours

1. Rappels sur les circuits synchrones
2. La sémantique par termes marqués (états)
3. Interface des circuits
4. Circuits de pause, abort, trap et suspend

Circuit d'une instruction, v0



E : signaux entrants

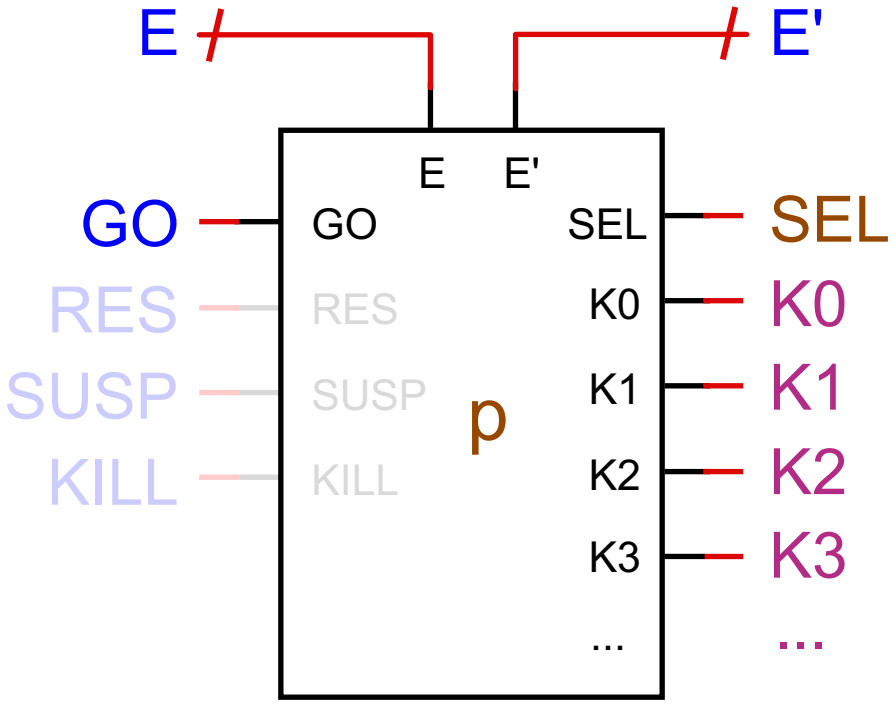
GO : démarrage de p

RES : reprise de p̂

SUSP : suspension de p̂

KILL : mort de p̂ à la fin de l'instant

Circuit d'une instruction, v0



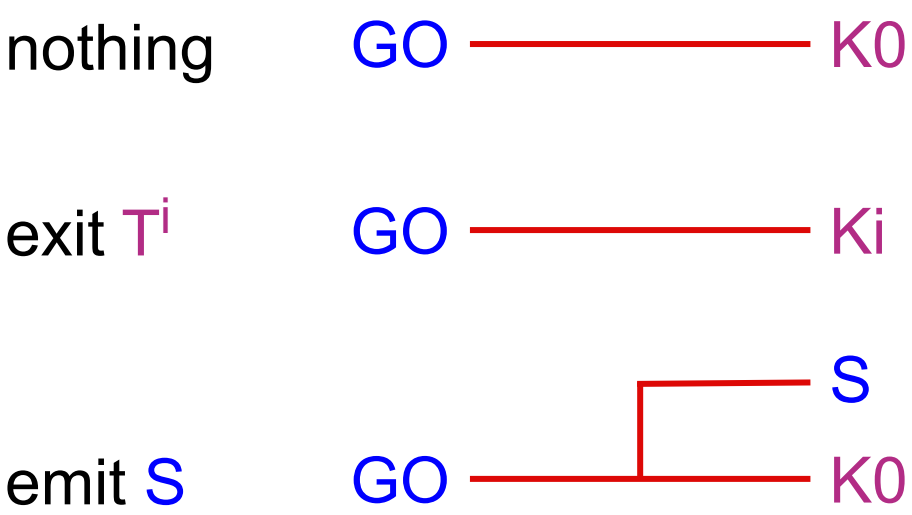
E' : signaux émis

SEL : sélection (activation), indique p si 0 et p̂ si 1

K0, K1, K2,… : codes de terminaison (*one-hot*)

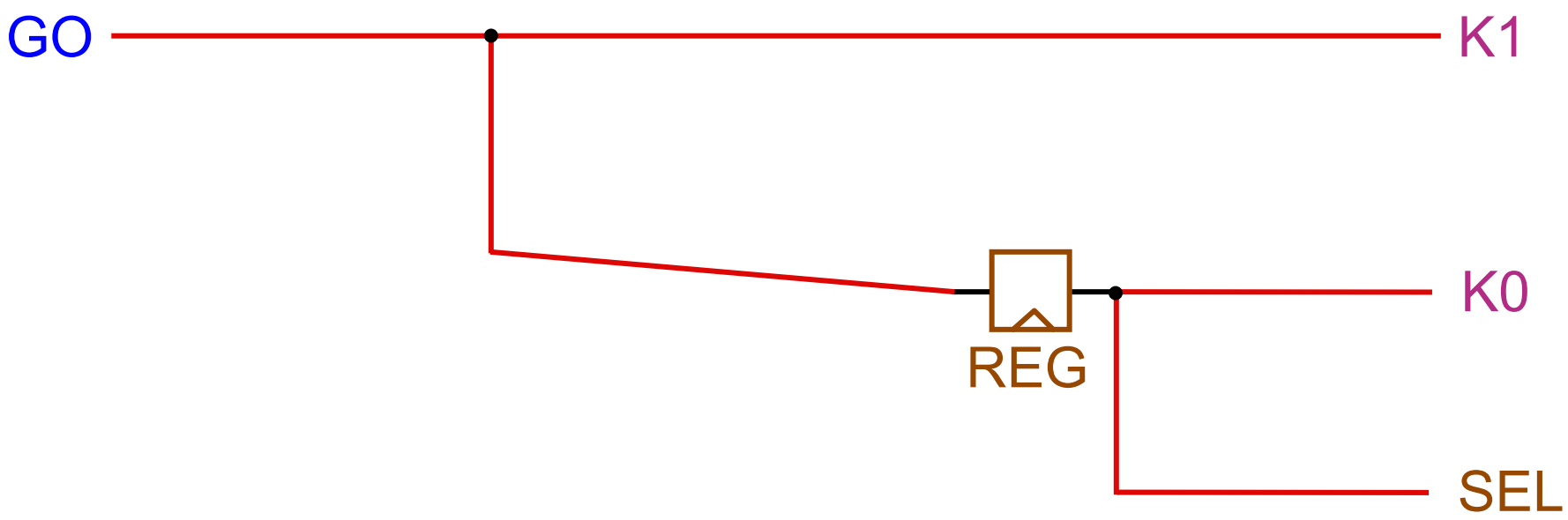
Fils mis à 0 par défaut si non mentionnés

Circuit de nothing, emit et exit



Circuit de la pause, v0

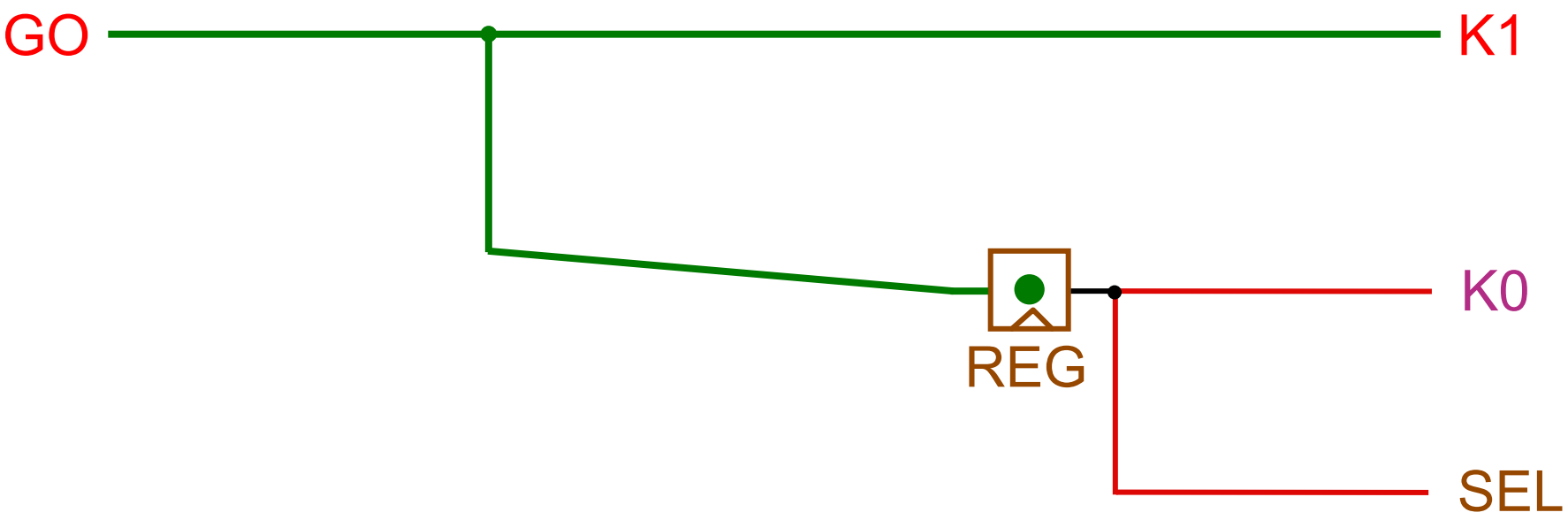
1990:
in GO, out K0, K1, SEL



Activation de la pause

1990:

go : REG=0 ; in GO=1 ; out K0=0, K1=1, SEL=0 ; REG←1

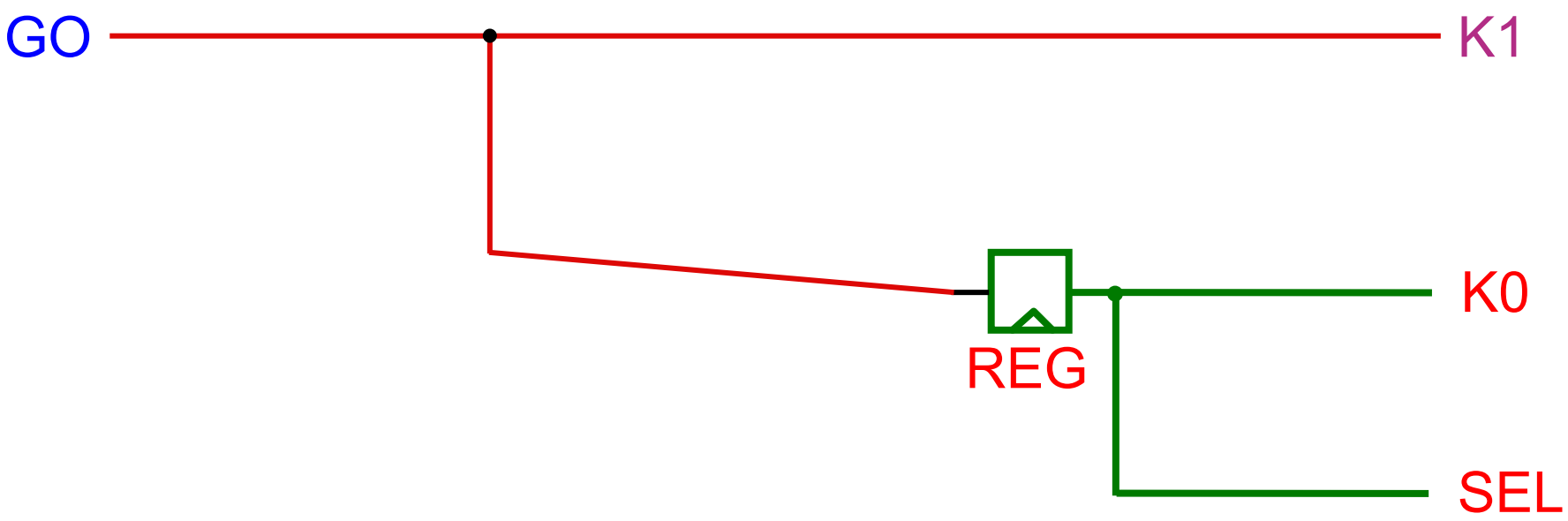


Reprise de la pause ⟹ terminaison, v0

1990:

go : REG=0 ; in GO=1 ; out K0=0, K1=1, SEL=0 ; REG←1

res : REG=1 ; in GO=0 ; out K0=SEL=1, K1=0; REG ←0



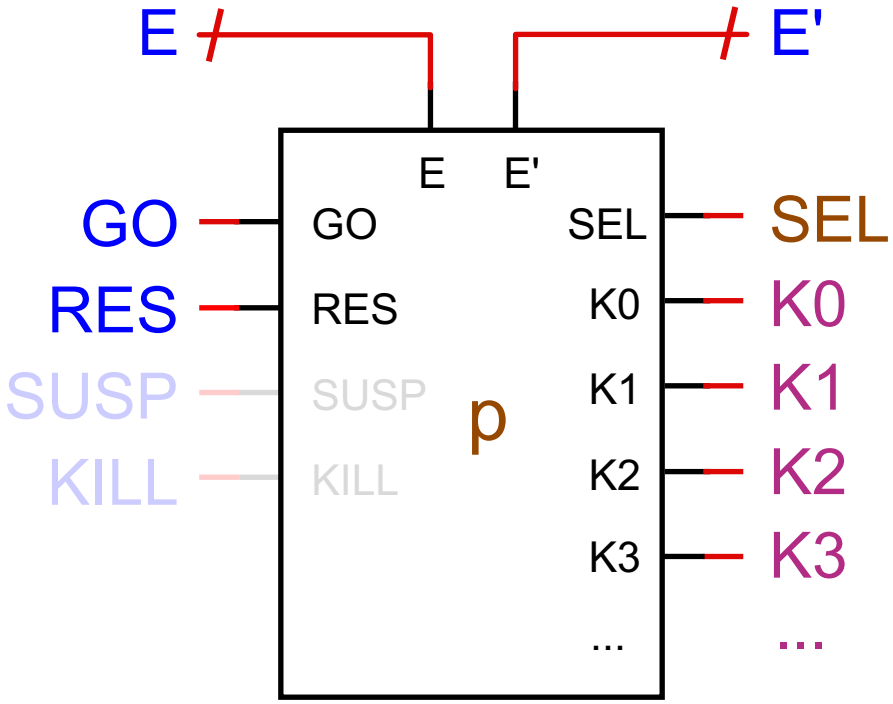
Mais abort contrôle la reprise

$$\frac{p \xrightarrow[I]{O,k} \overline{p}'}{\texttt{abort } p \texttt{ when } S \xrightarrow[I]{O,k} \texttt{abort } \overline{p}' \texttt{ when } S} \quad (go\text{-}abort)$$

$$\frac{S \notin I \qquad \hat{p} \xrightarrow[I]{O,k} \overline{p}'}{\texttt{abort } \hat{p} \texttt{ when } S \xrightarrow[I]{O,k} \texttt{abort } \overline{p}' \texttt{ when } S} \quad (res\text{-}abort-)$$

$$\frac{S \in I}{\texttt{abort } \hat{p} \texttt{ when } S \xrightarrow[I]{\emptyset,0} \texttt{abort } p \texttt{ when } S} \quad (res\text{-}abort+)$$

Contrôle de la reprise ⟹ RES



SEL=RES=1 : reprise de p̂

SEL=1, RES=0 : abort fort de p̂

SEL=0 : RES=0 ou RES=1 n'ont aucun effet sur p

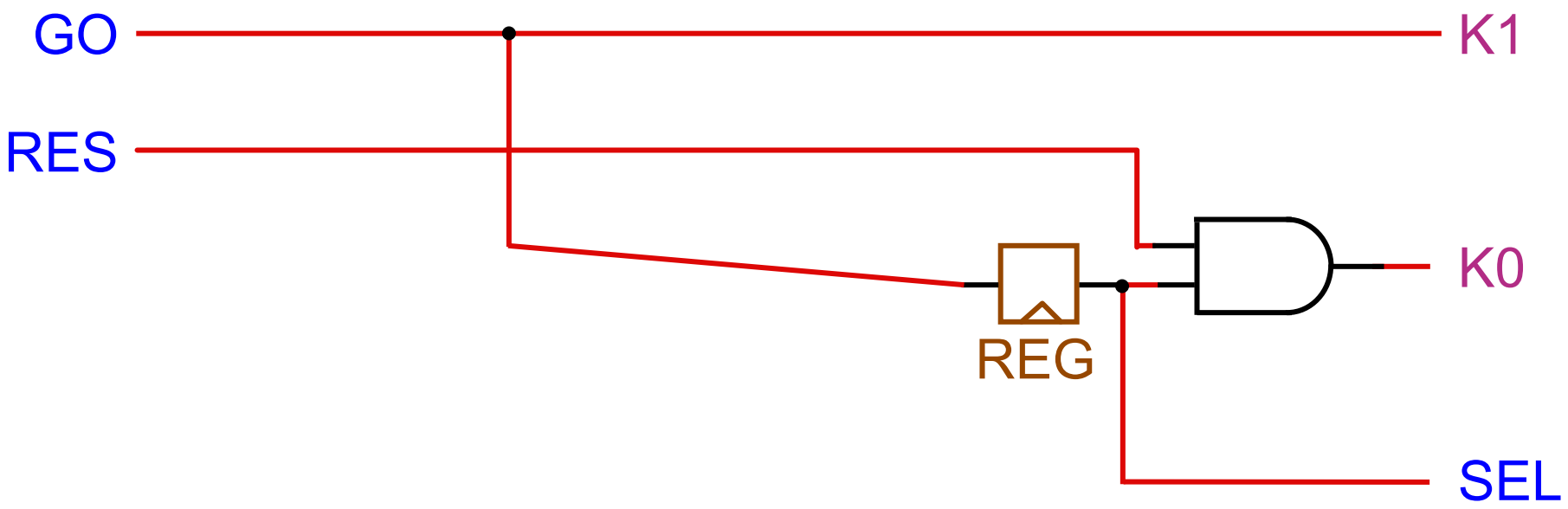
Circuit de pause, v1

1990:

go : REG=0 ; in GO=1, RES=0 ; out K0=0, K1=1, SEL=0 ; REG←1

res : terminer si activé ; in RES=1

abort : mourir sans terminer ; in RES=0



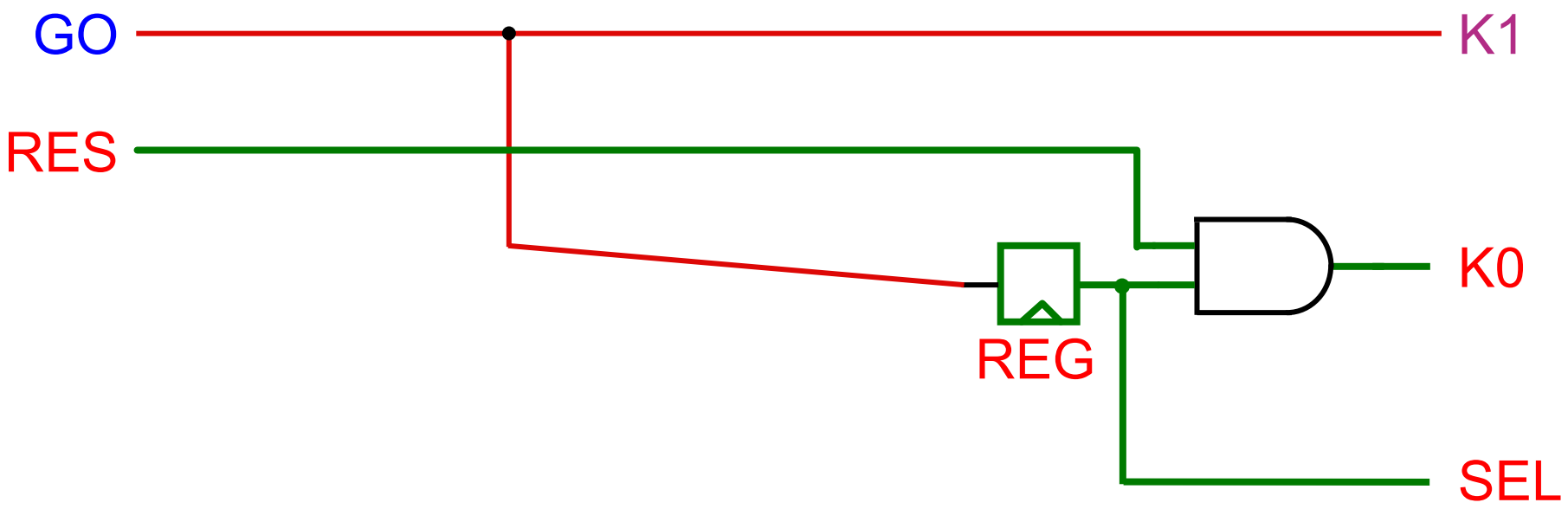
RES et REG à 1 ⟹ terminaison

1990:

go : REG=0 ; in GO=1, RES=0 ; out K0=0, K1=1, SEL=0 ; REG←1

res : REG=1 ; in GO=0, RES=1 ; out K0=SEL=1, K1=0 ; REG ←0

abort fort : entrée RES=0



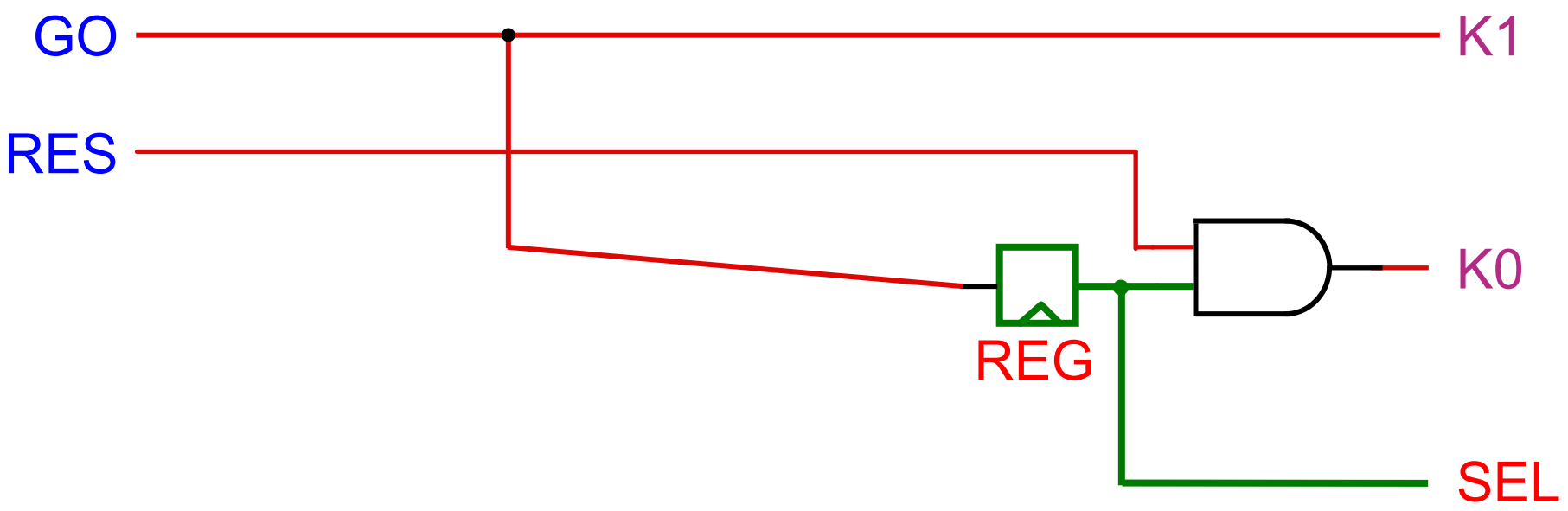
RES=0 ⟹ abort du pause sans terminaison

1990:

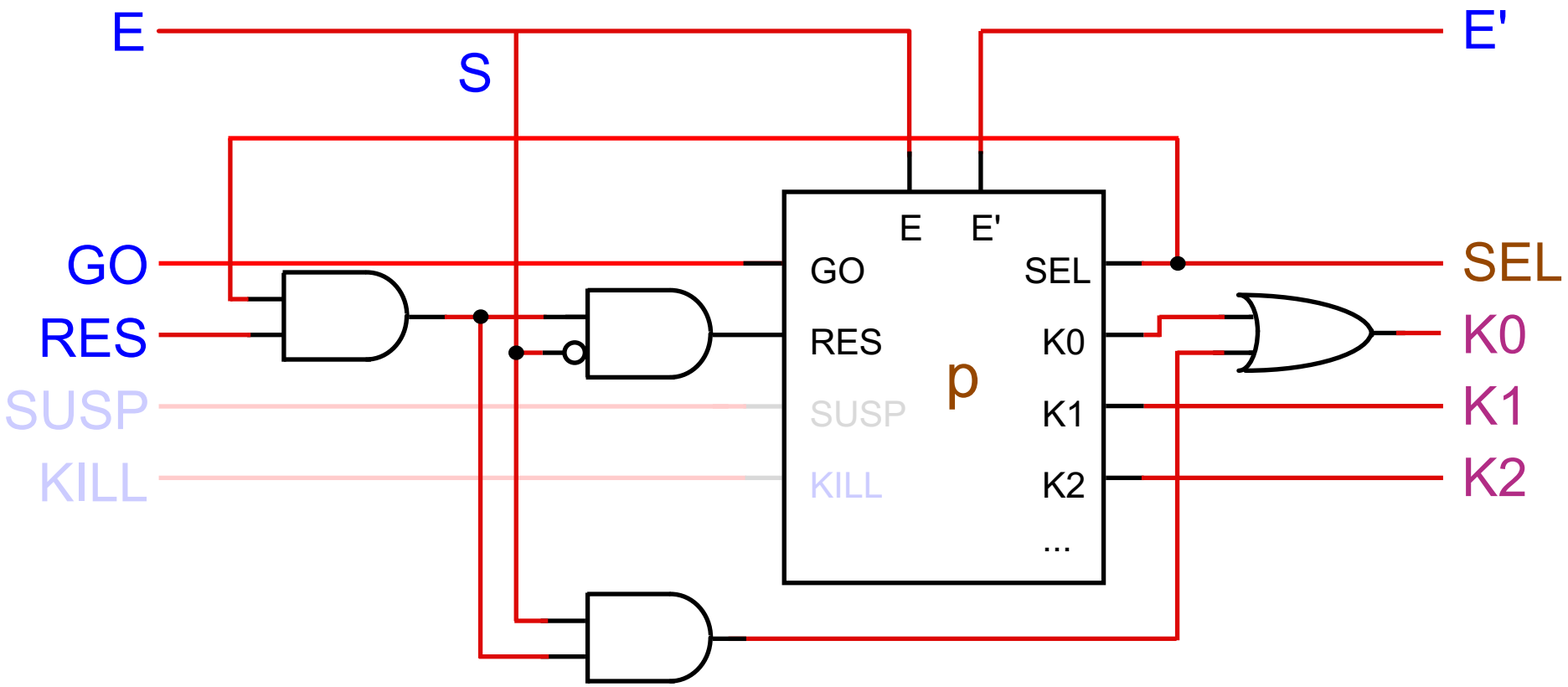
go : REG=0 ; in GO=1, RES=0 ; out K0=0, K1=1, SEL=0 ; REG←1

res : REG=1 ; in GO=0, RES=1 ; out K0=SEL=1, K1=0 ; REG ←0

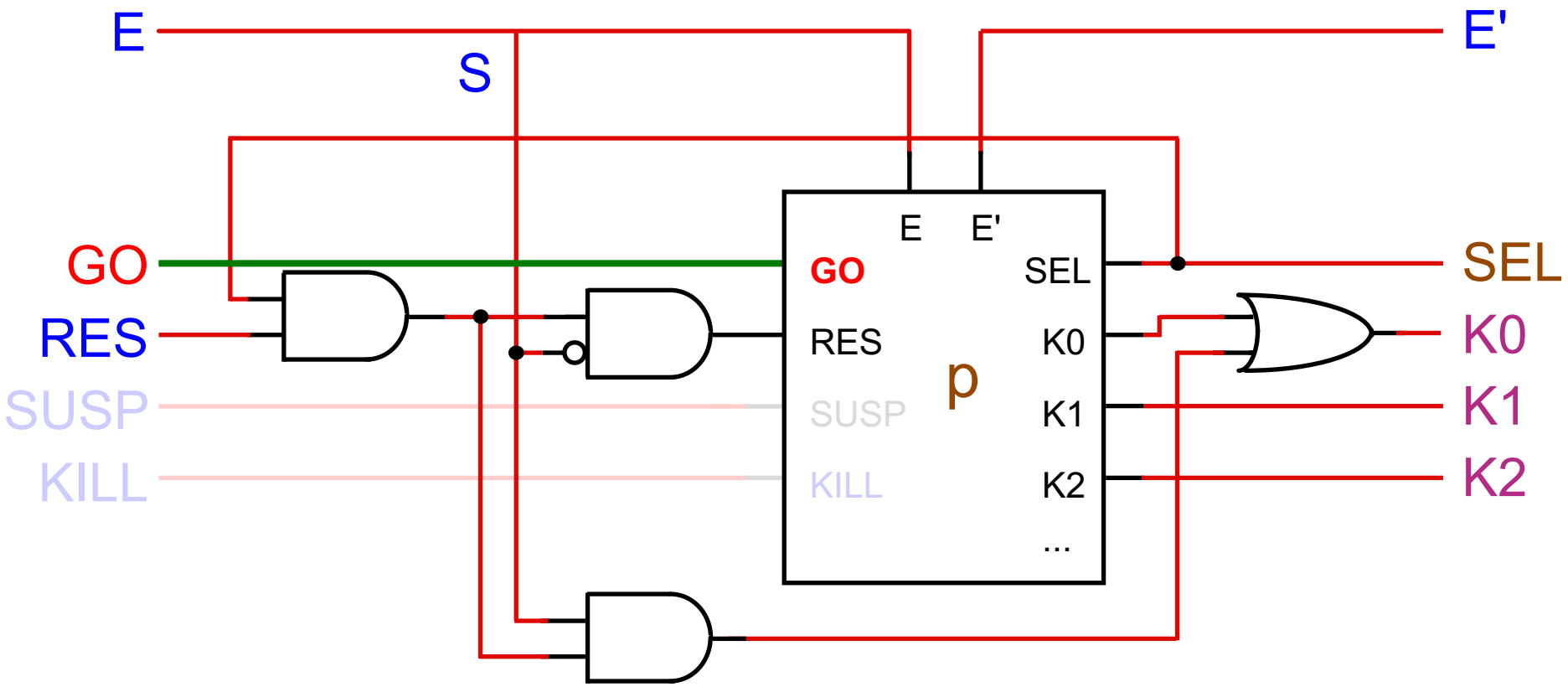
abort : REG=1 ; in GO=RES=0 ; out K0=K1=0, SEL=1 ; REG←0



Circuit pour « abort p when S »

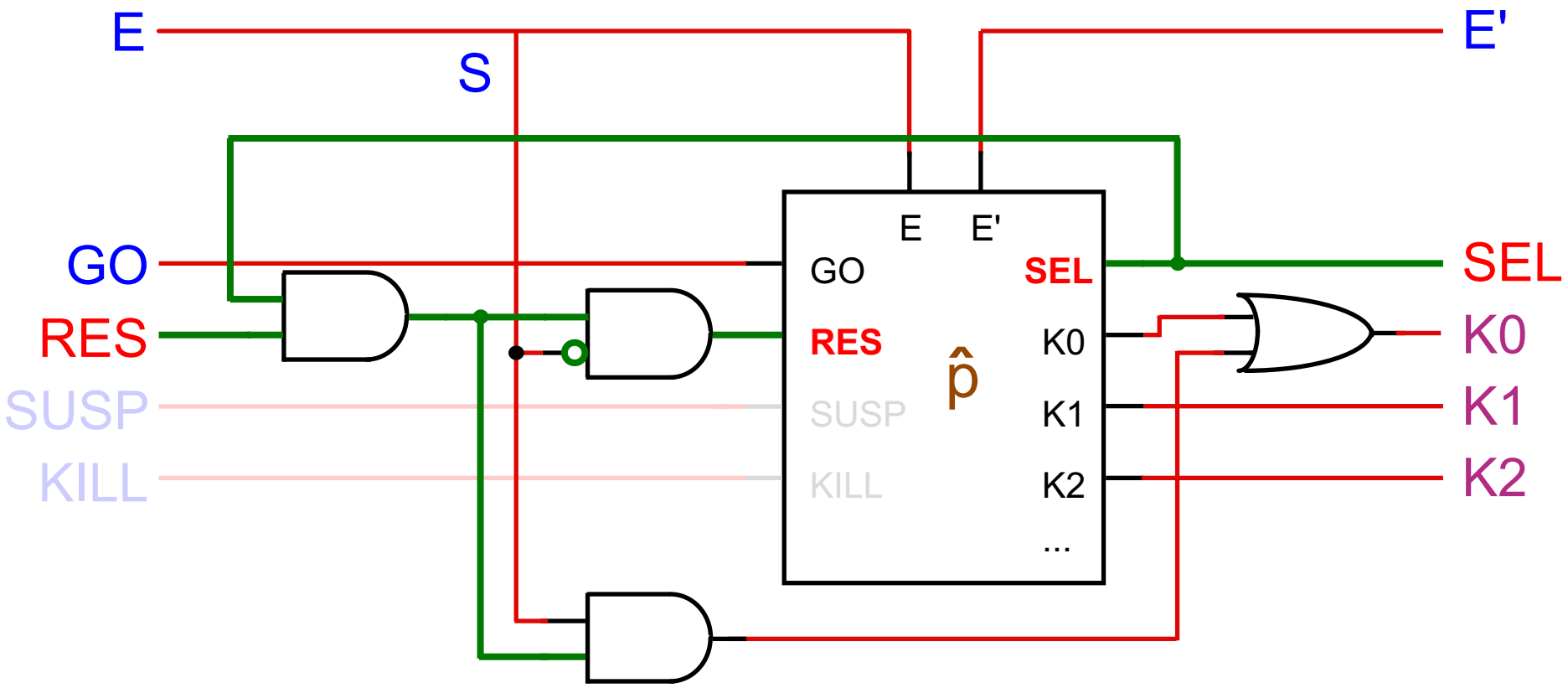


Circuit pour « abort p when S », go



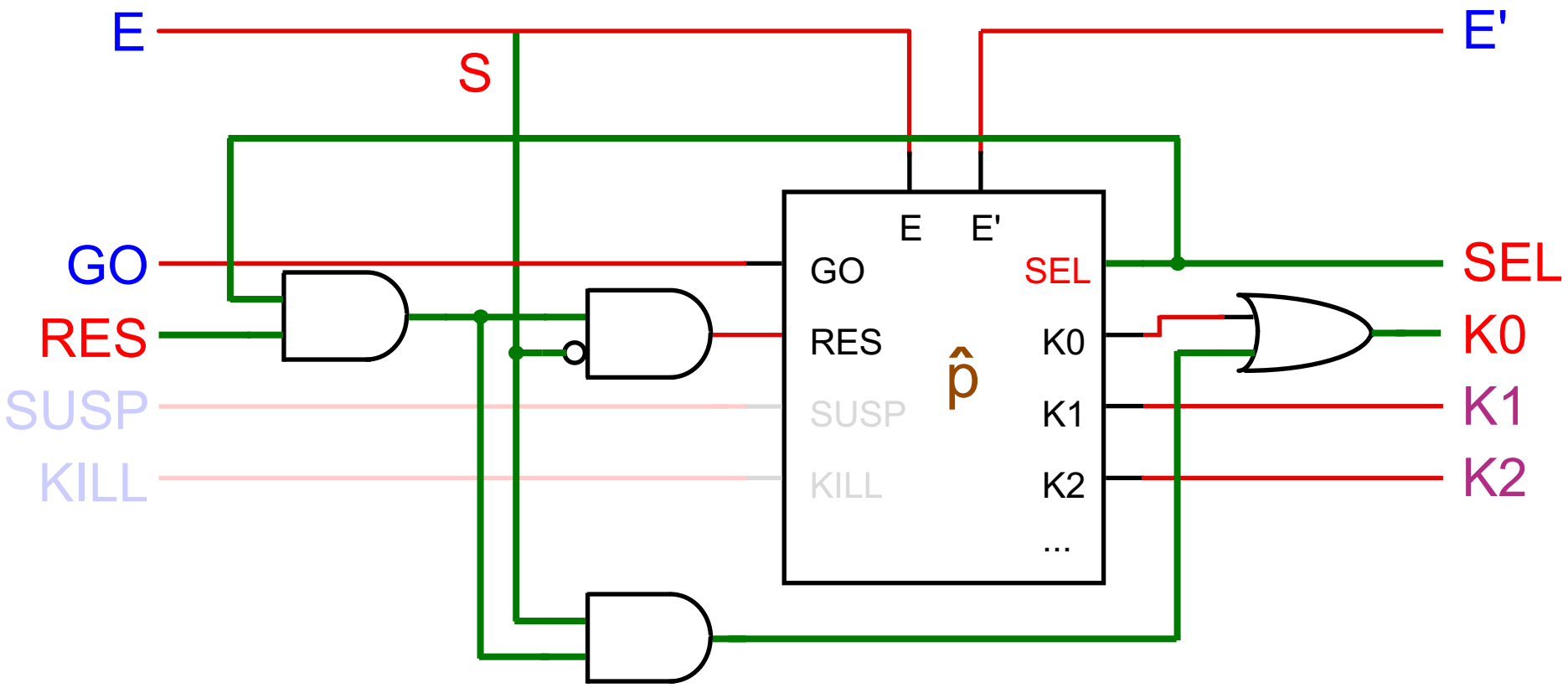
abort retardé ⟹ GO est directement transmis à p

Circuit pour « abort p̂ when S », S absent



RES est transmis à p̂, qui mettra un des Ki à 1

Circuit pour « abort p̂ when S », S présent



RES n’est pas transmis à p̂, K0 est mis à 1

Mais trap contrôle la prémption faible

$$\frac{\overline{p} \xrightarrow[I]{O,\, k} \overline{p}'}{\texttt{trap}\ T\ \texttt{in}\ \overline{p}\ \texttt{end} \xrightarrow[I]{O,\, \downarrow k} \texttt{trap}\ T\ \texttt{in}\ \overline{p}'\ \texttt{end}} \quad (go\text{-}res\text{-}trap)$$

$$\frac{p \xrightarrow[I]{O,\, k} \overline{p}'}{\texttt{trap}\ T\ \texttt{in}\ p\ \texttt{end} \xrightarrow[I]{O,\, \downarrow k} \texttt{trap}\ T\ \texttt{in}\ \overline{p}'\ \texttt{end}} \quad (go\text{-}trap)$$

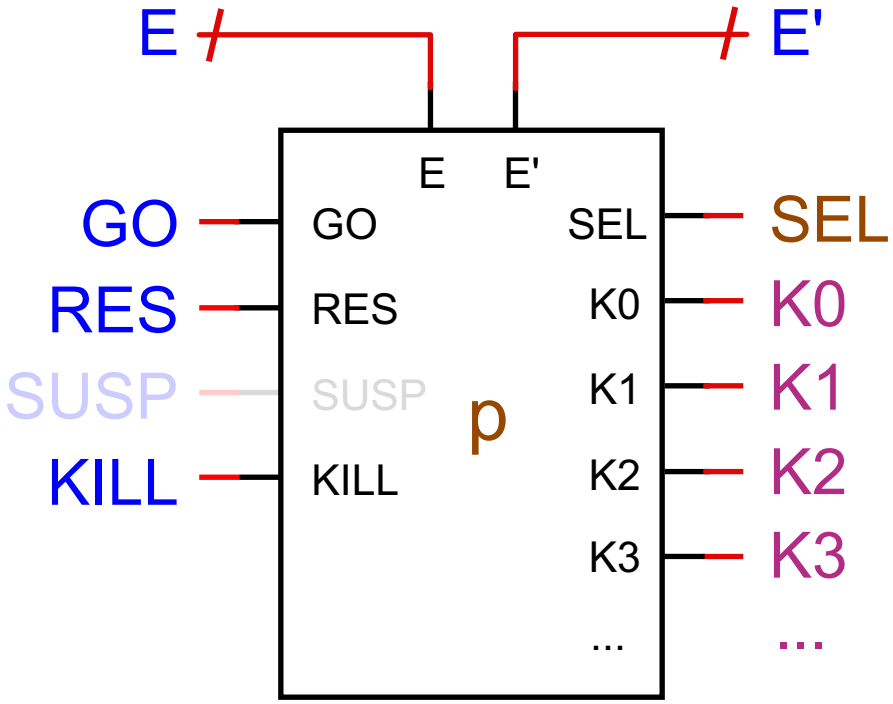
$$\frac{\hat{p} \xrightarrow[I]{O,\, k} \overline{p}'}{\texttt{trap}\ T\ \texttt{in}\ \hat{p}\ \texttt{end} \xrightarrow[I]{O,\, \downarrow k} \texttt{trap}\ T\ \texttt{in}\ \overline{p}'\ \texttt{end}} \quad (res\text{-}trap)$$

Le cas intéressant : sortie de cette trappe, k=2

$$\frac{\overline{p} \xrightarrow[I]{O,2} \overline{p}'}{\texttt{trap } T \texttt{ in } \overline{p} \texttt{ end} \xrightarrow[I]{O,0} \texttt{trap } T \texttt{ in } p \texttt{ end}} \quad (go\text{-}res\text{-}trap\text{-}2)$$

Tous les pause sont transformés en pause

Contrôle de la préemption faible ⟹ KILL



GO=KILL=1, SEL=0 : p démarre normalement, mais meurt

RES= KILL=1, SEL=1 : p̂ agit normalement, mais meurt

Circuit de pause, v2

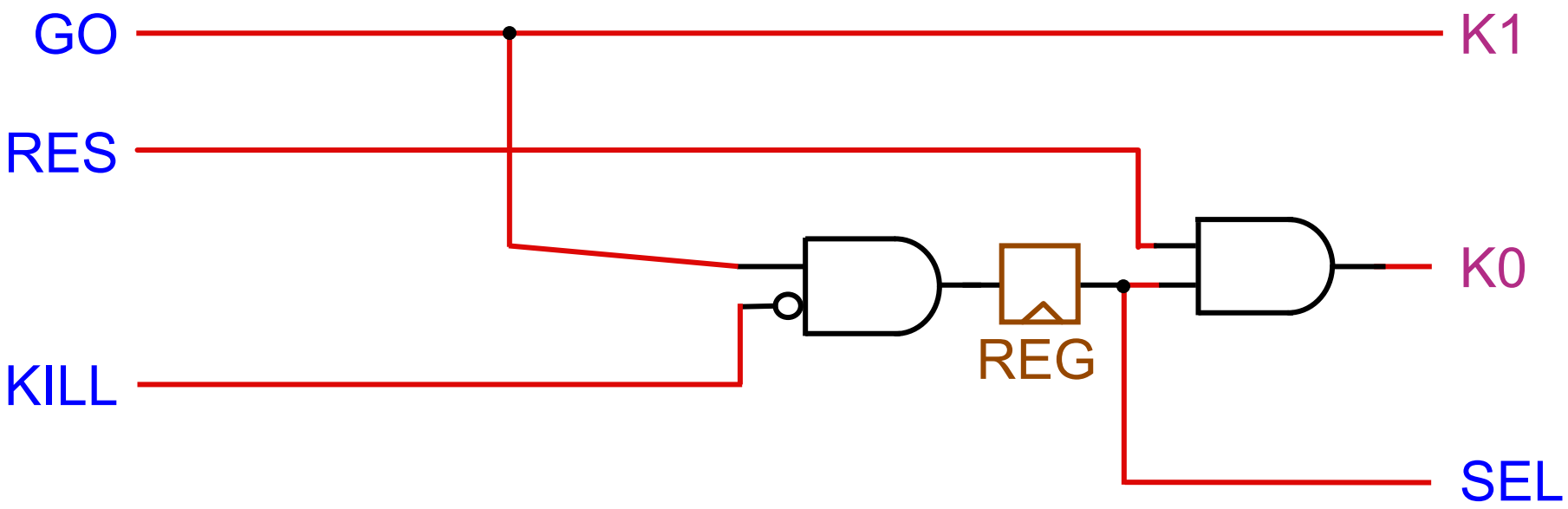
1990:

go : REG=0 ; in GO=1, RES=0 ; out K0=0, K1=1, SEL=0 ; REG←1

res : REG=1 ; in GO=0, RES=1 ; out K0=SEL=1, K1=0 ; REG ←0

abort : REG=1 ; in GO=RES=0 ; out K0=K1=0, SEL=1 ; REG←0

kill : entrée KILL pour la préemption faible



Circuit de pause, v2

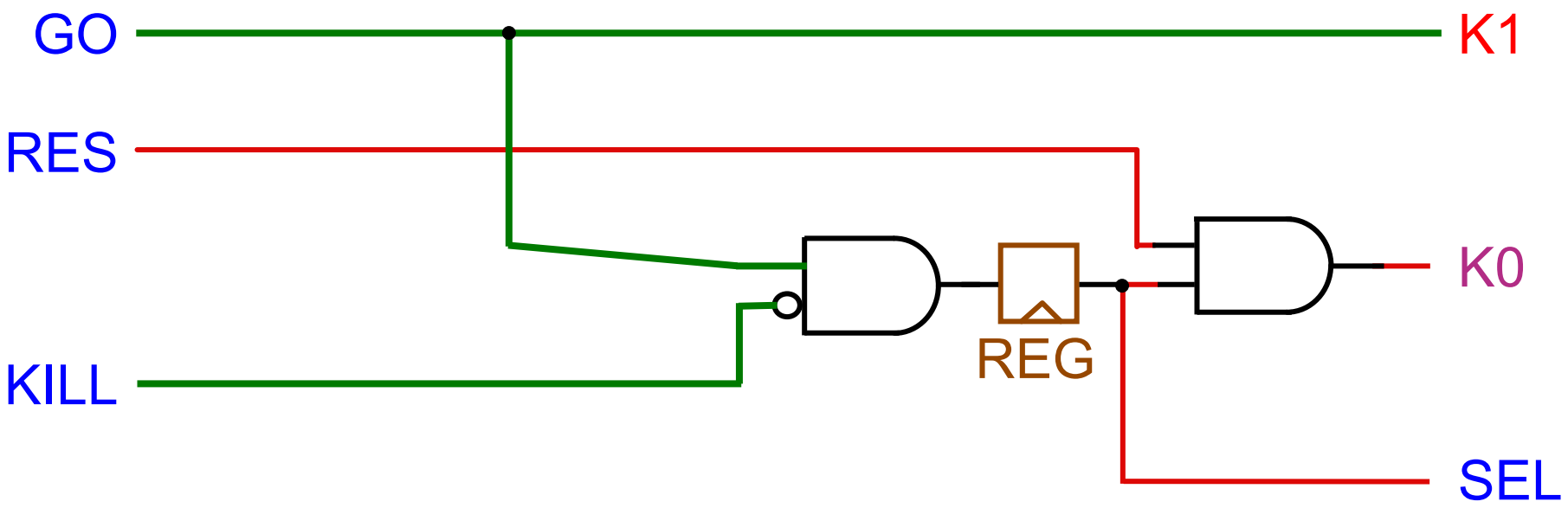
1990:

go : REG=0 ; in GO=1, RES=0 ; out K0=0, K1=1, SEL=0 ; REG←1

res : REG=1 ; in GO=0, RES=1 ; out K0=SEL=1, K1=0 ; REG ←0

abort : REG=1 ; in GO=RES=0 ; out K0=K1=0, SEL=1 ; REG←0

go-kill : REG=0 ; in GO=KILL=1, RES=0; out K1=1, K0=SEL=0 ; REG ←0



Circuit de pause, v2

1990:

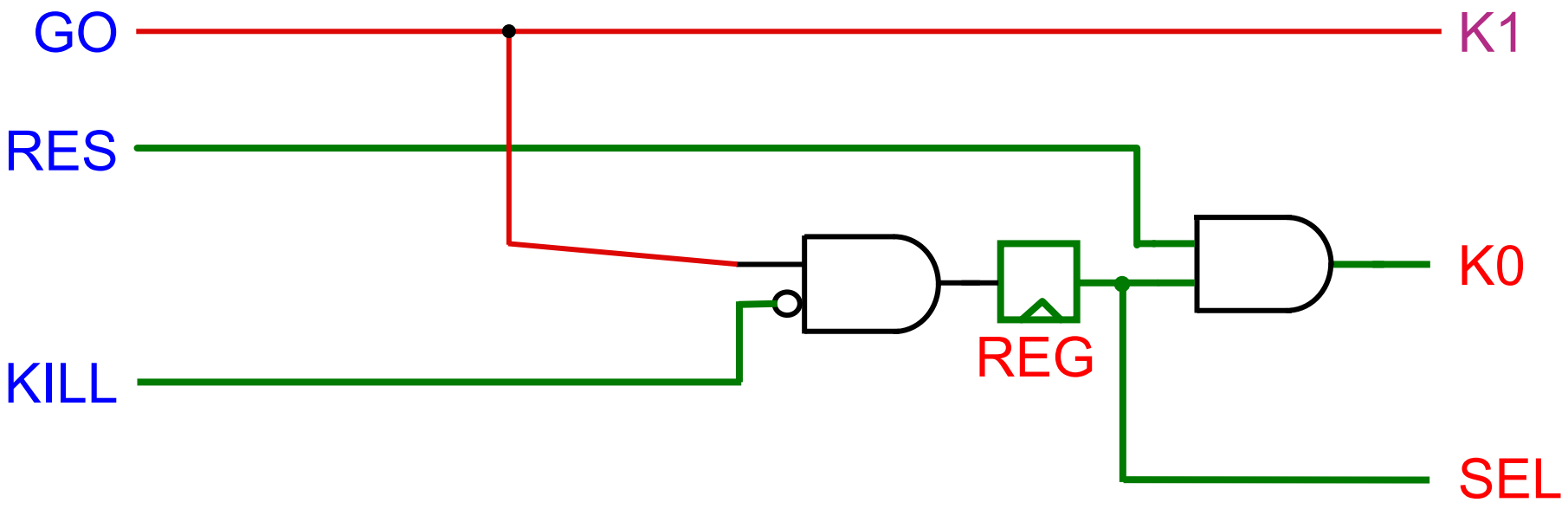
go : REG=0 ; in GO=1, RES=0 ; out K0=0, K1=1, SEL=0 ; REG←1

res : REG=1 ; in GO=0, RES=1 ; out K0=SEL=1, K1=0 ; REG ←0

abort : REG=1 ; in GO=RES=0 ; out K0=K1=0, SEL=1 ; REG←0

go-kill : REG=0 ; in GO=KILL=1, RES=0; out K1=1, K0=SEL=0 ; REG ←0

res-kill : REG=1 ;in G0=0, RES=KILL=1 ; out K1=0, K0=SEL=1; REG←0



Comportement complet du pause v2

1990:

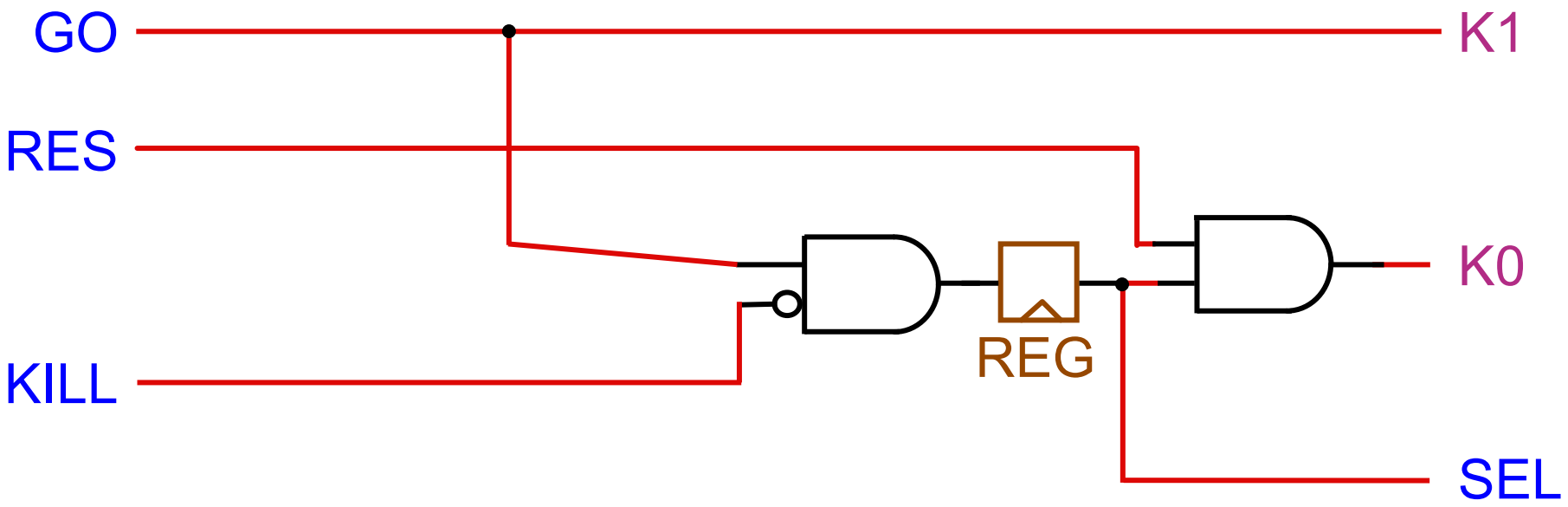
go : REG=0 ; in GO=1, RES=0 ; out K0=0, K1=1, SEL=0 ; REG←1

res : REG=1 ; in GO=0, RES=1 ; out K0=SEL=1, K1=0 ; REG ←0

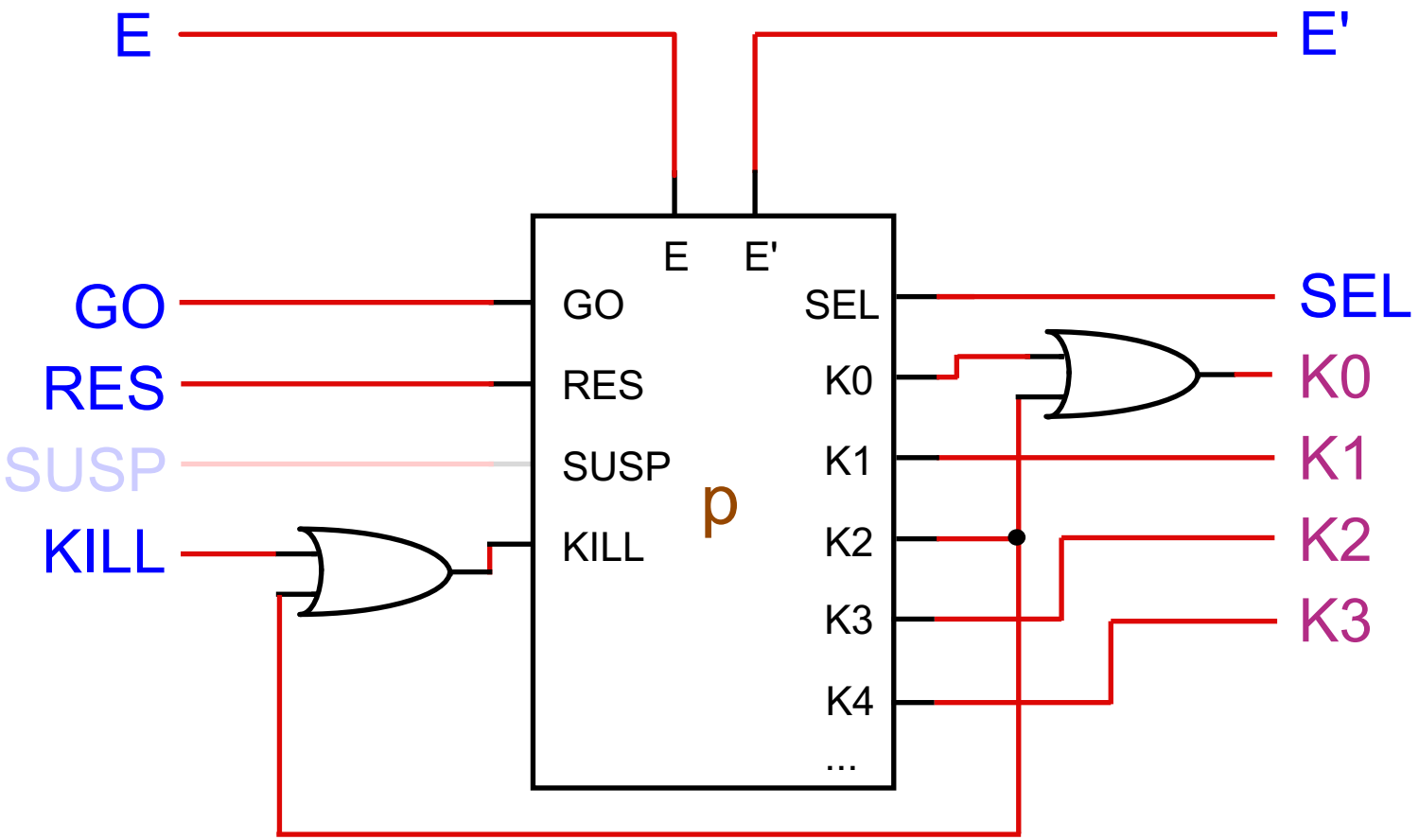
abort : REG=1 ; in GO=RES=0 ; out K0=K1=0, SEL=1 ; REG←0

go-kill : REG=0 ; in GO=KILL=1, RES=0; out K1=1, K0=SEL=0 ; REG ←0

res-kill : REG=1 ;in G0=0, RES=KILL=1 ; out K1=0, K0=SEL=1; REG←0

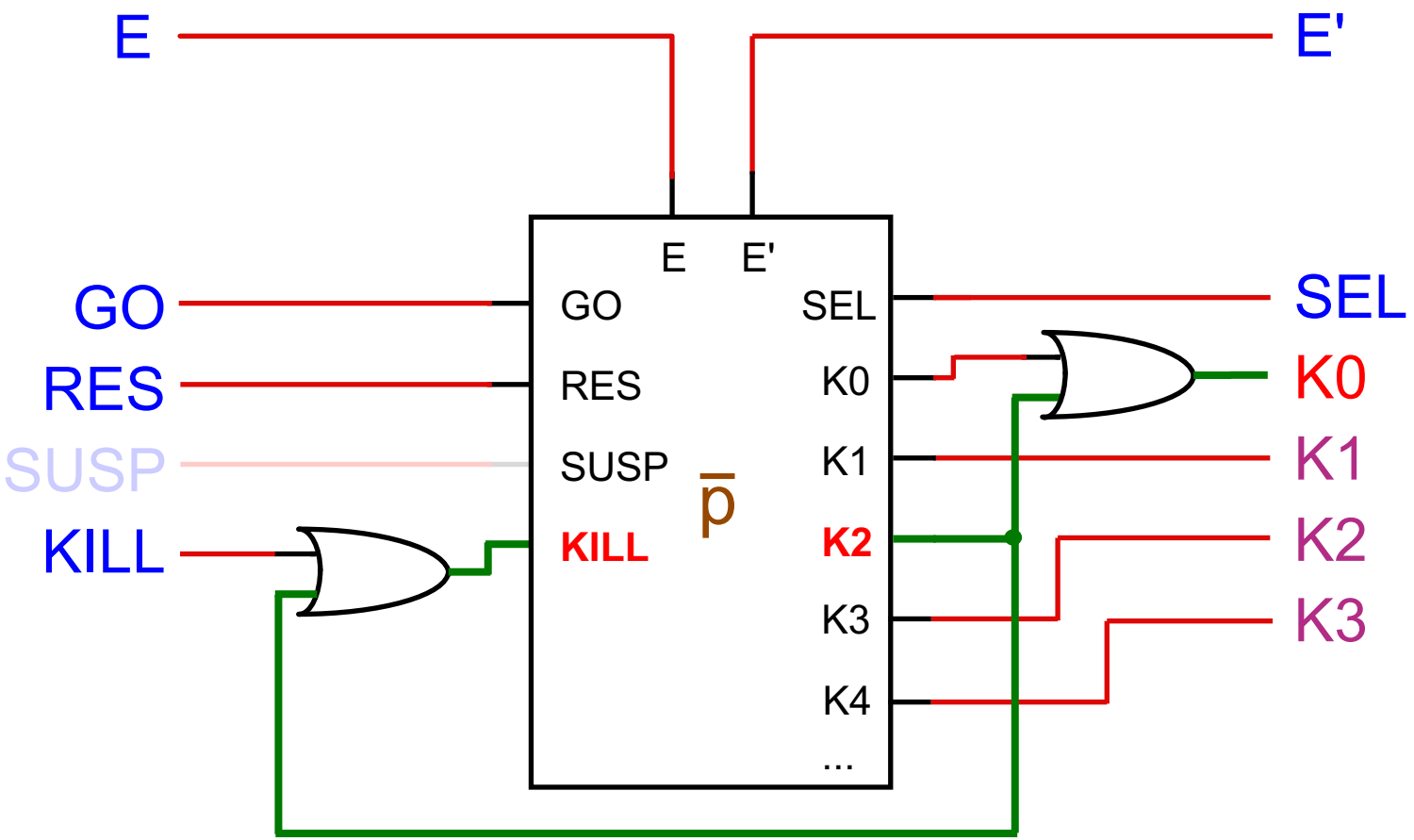


Circuit pour « trap T in p end »



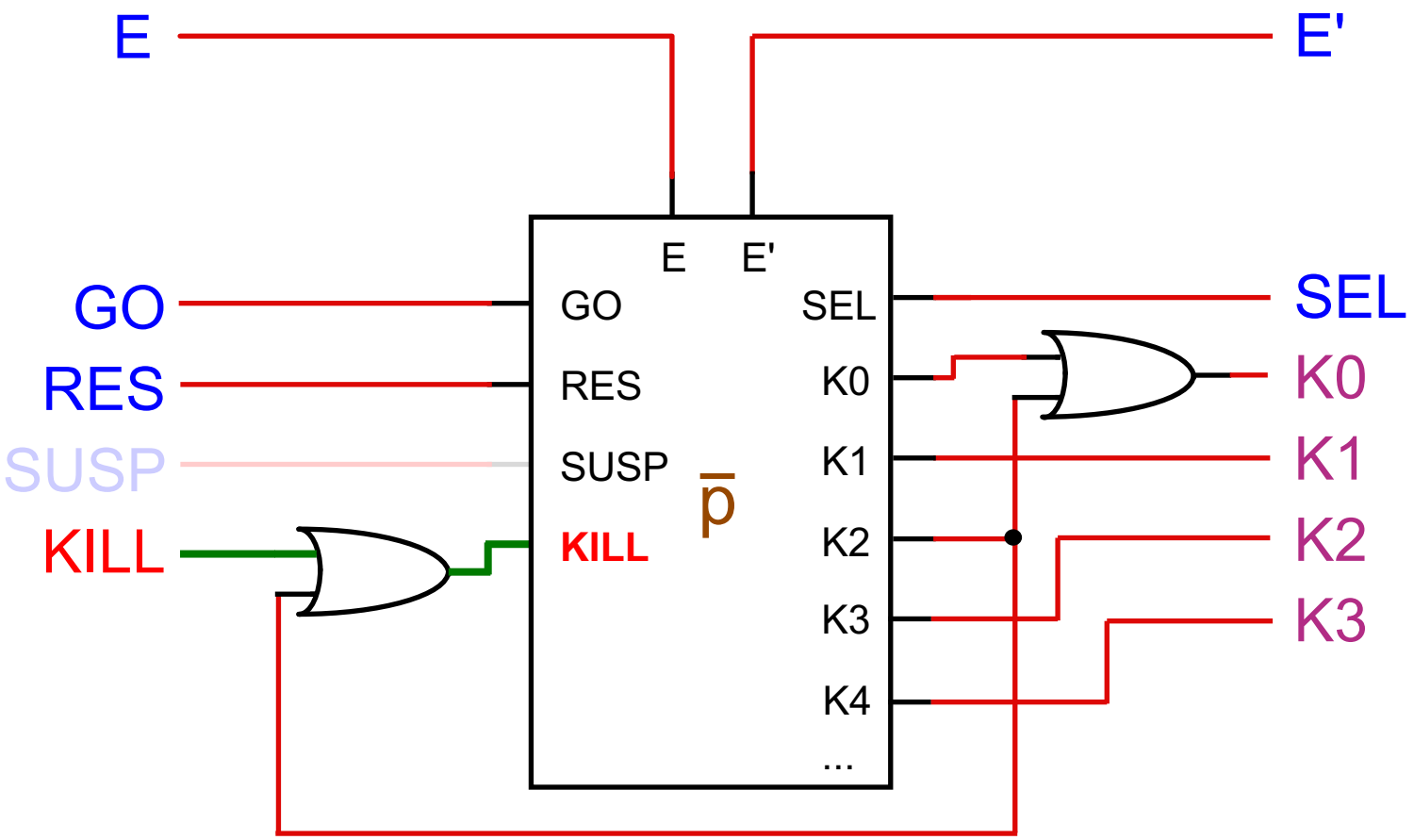
K2 devient K0 et revient dans KILL, K1 inchangé
les autres décalés d'un cran vers le bas (↓k)

Circuit pour « trap T in p end »



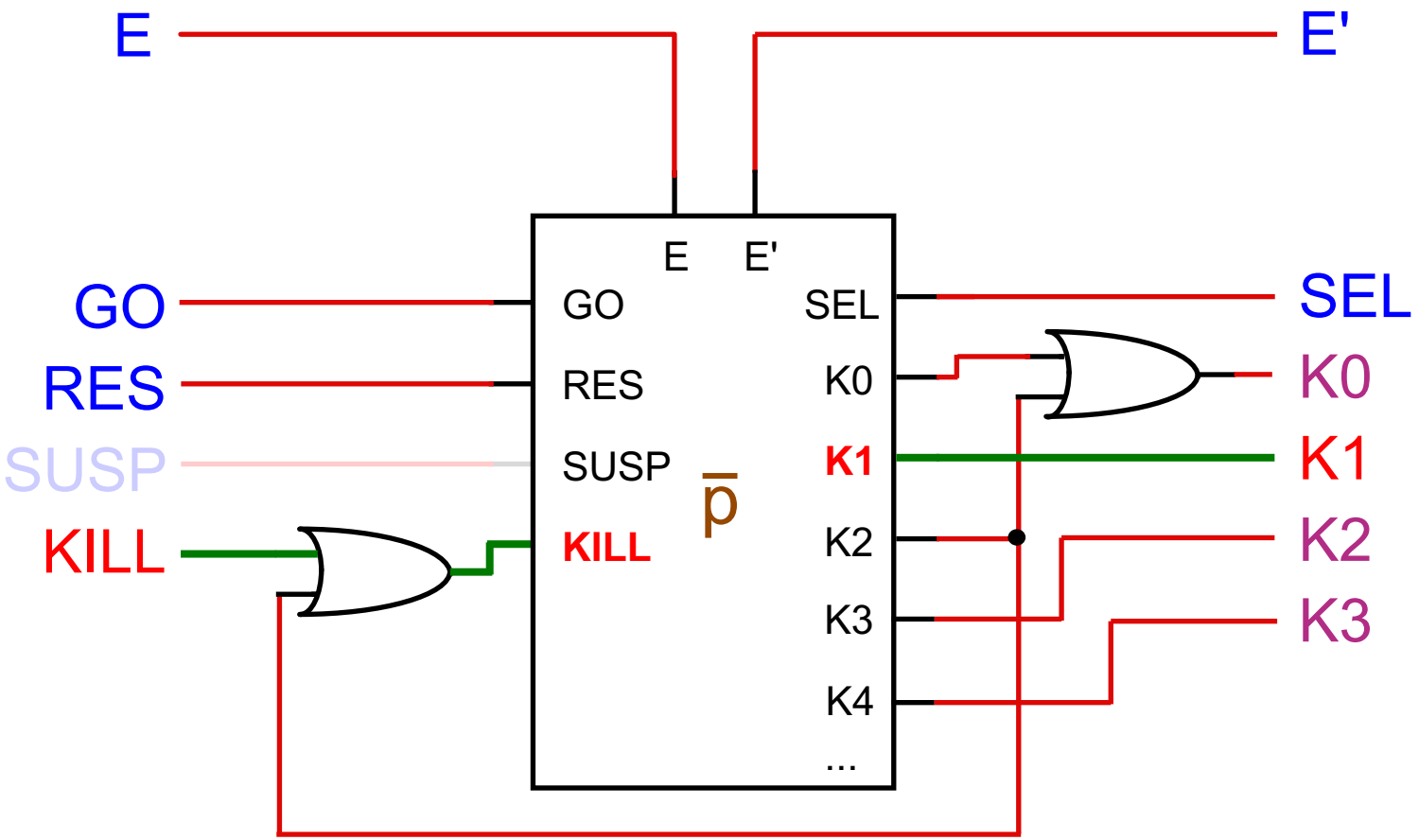
Si p̄ rend K2=1, terminaison de cette trappe avec K0=1
et mort de p̄ par KILL

Circuit pour « trap T in p end »

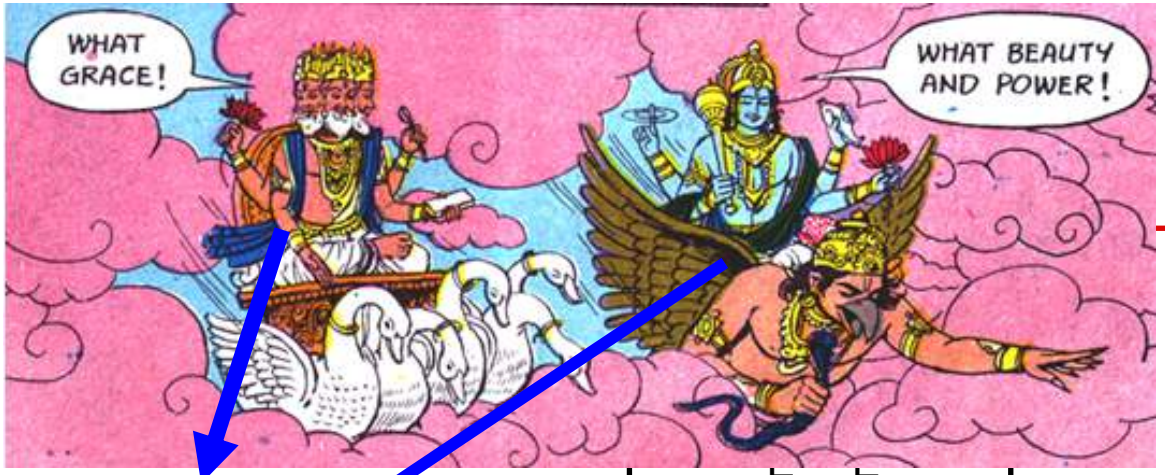


Un KILL venant d'une trappe englobante est propagé mais $\bar{p}$ s'exécute normalement avant de mourir, avec un Ki à 1

Circuit pour « trap T in p end »



Un KILL venant d'une trappe englobante est propagé mais p̄ s'exécute mormalement avant de mourir, avec un Ki à 1



E'

GO

RES

SUSP

KILL

GO

RES

SUSP

KILL

SEL

SEL

p

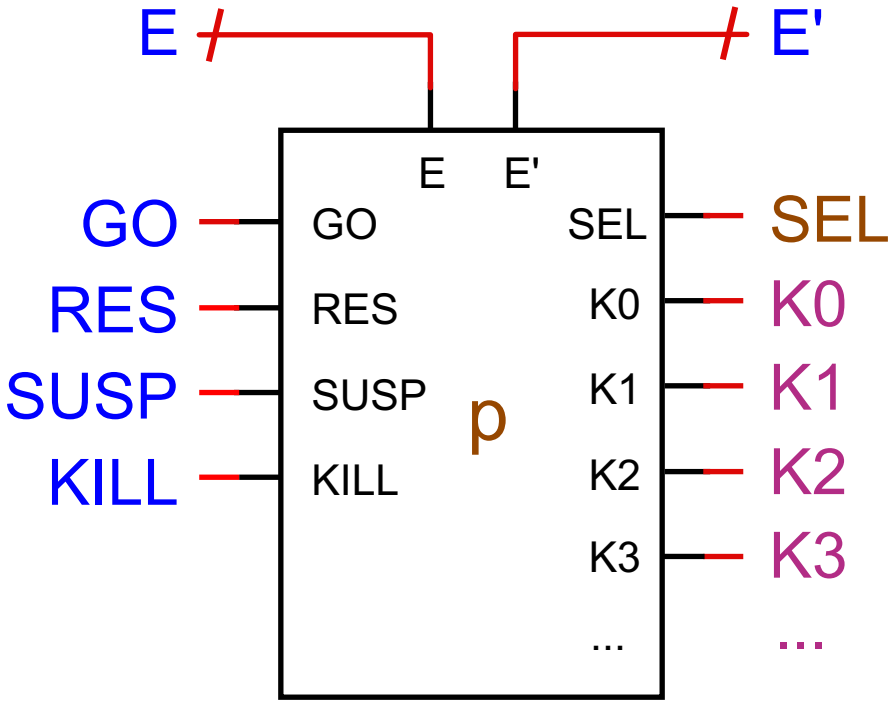
1995 : suspend contrôle la suspension

$$\frac{p \xrightarrow[I]{O,k} \overline{p}'}{\texttt{suspend}\ p\ \texttt{when}\ S \xrightarrow[I]{O,k} \texttt{suspend}\ \overline{p}'\ \texttt{when}\ S} \quad (go\text{-}suspend)$$

$$\frac{S \notin I \qquad \hat{p} \xrightarrow[I]{O,k} \overline{p}'}{\texttt{suspend}\ \hat{p}\ \texttt{when}\ S \xrightarrow[I]{O,k} \texttt{suspend}\ \overline{p}'\ \texttt{when}\ S} \quad (res\text{-}suspend{-})$$

$$\frac{S \in I}{\texttt{suspend}\ \hat{p}\ \texttt{when}\ S \xrightarrow[I]{\emptyset,1} \texttt{suspend}\ \hat{p}\ \texttt{when}\ S} \quad (res\text{-}suspend{+})$$

Contrôle de la suspension ⟹ SUSP



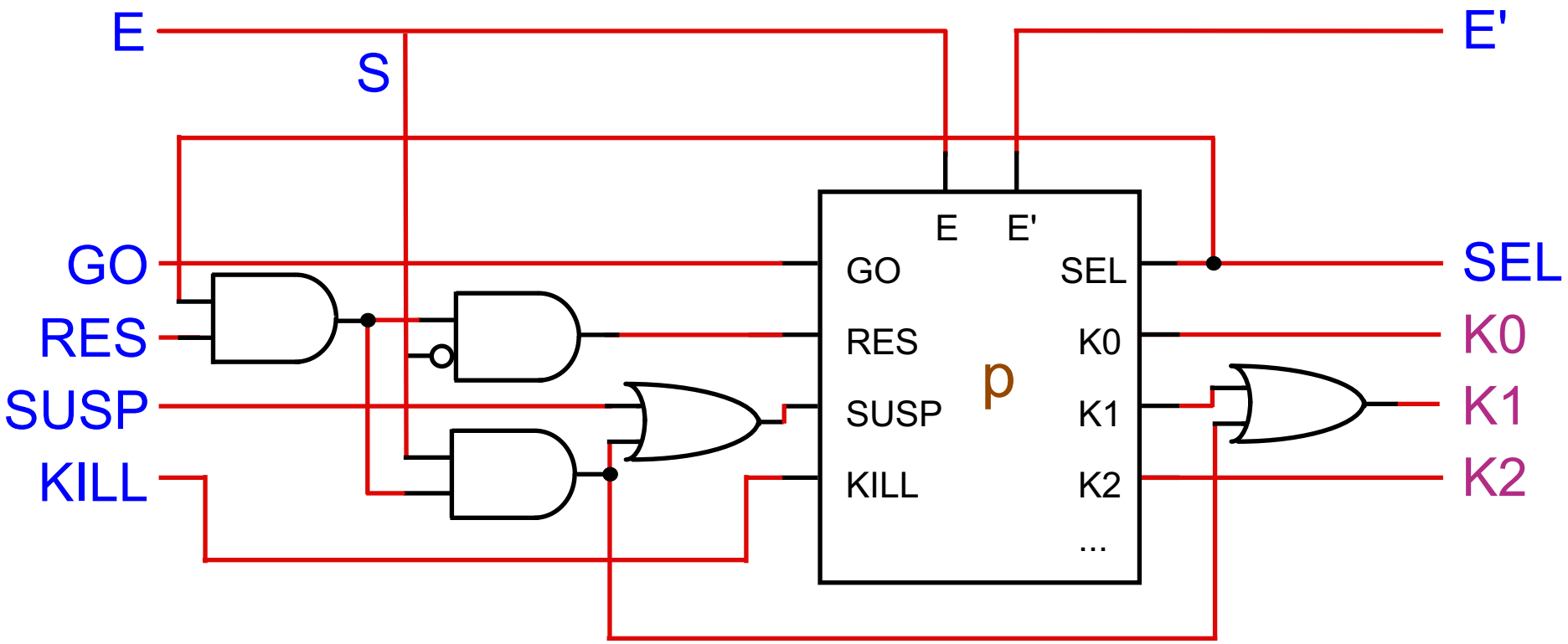
RES=SEL=1, SUSP=0 : reprise de p

SEL=1, RES=0, SUSP=1 : suspension de p̂

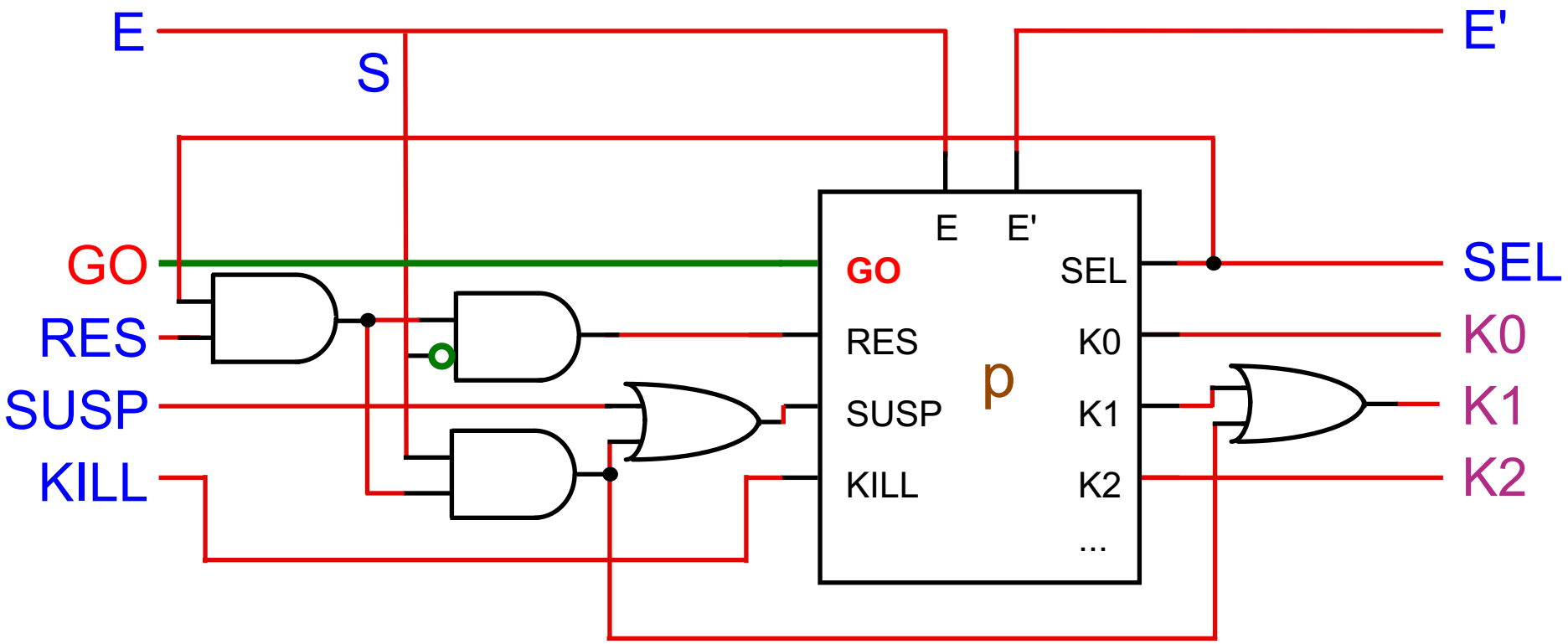
Note : pour l'instant, RES et SUSP sont incompatibles

RES # SUSP *i.e.* ¬(RES and SUSP)

Circuit de « suspend p when S »

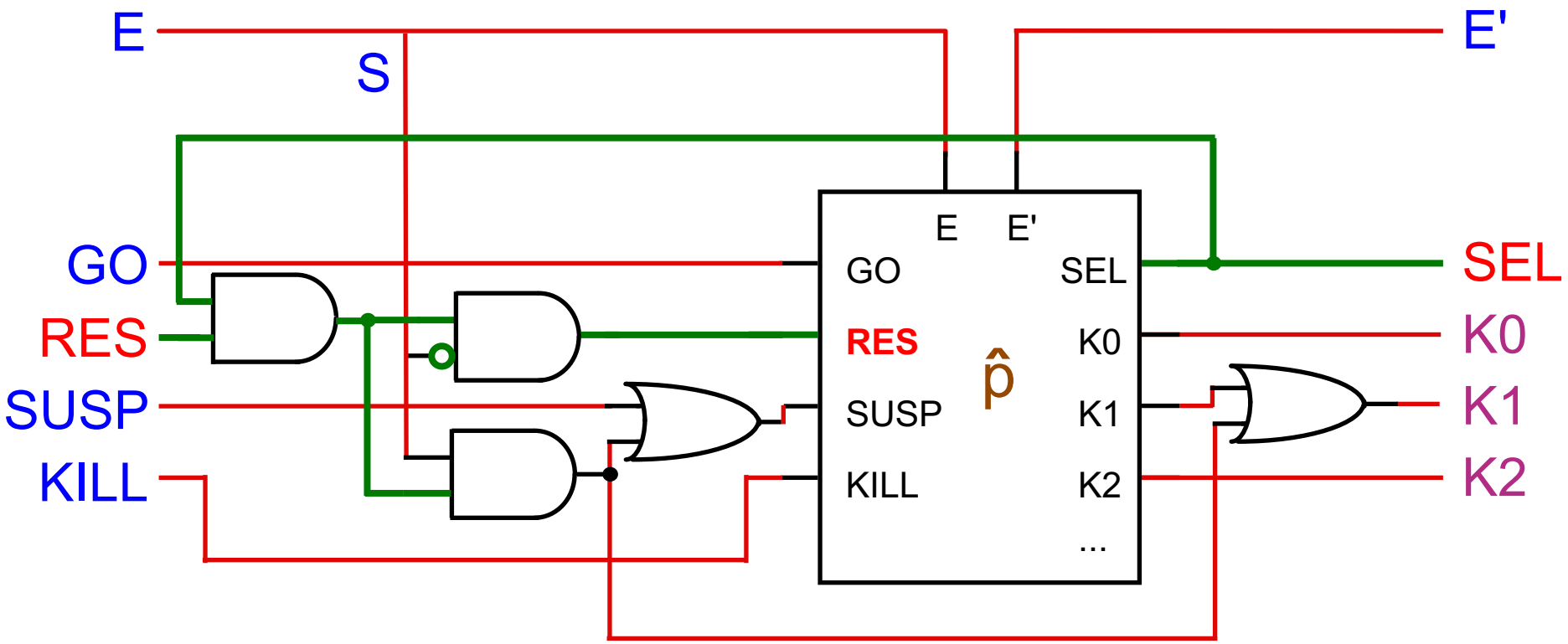


Circuit de « suspend p when S », démarrage



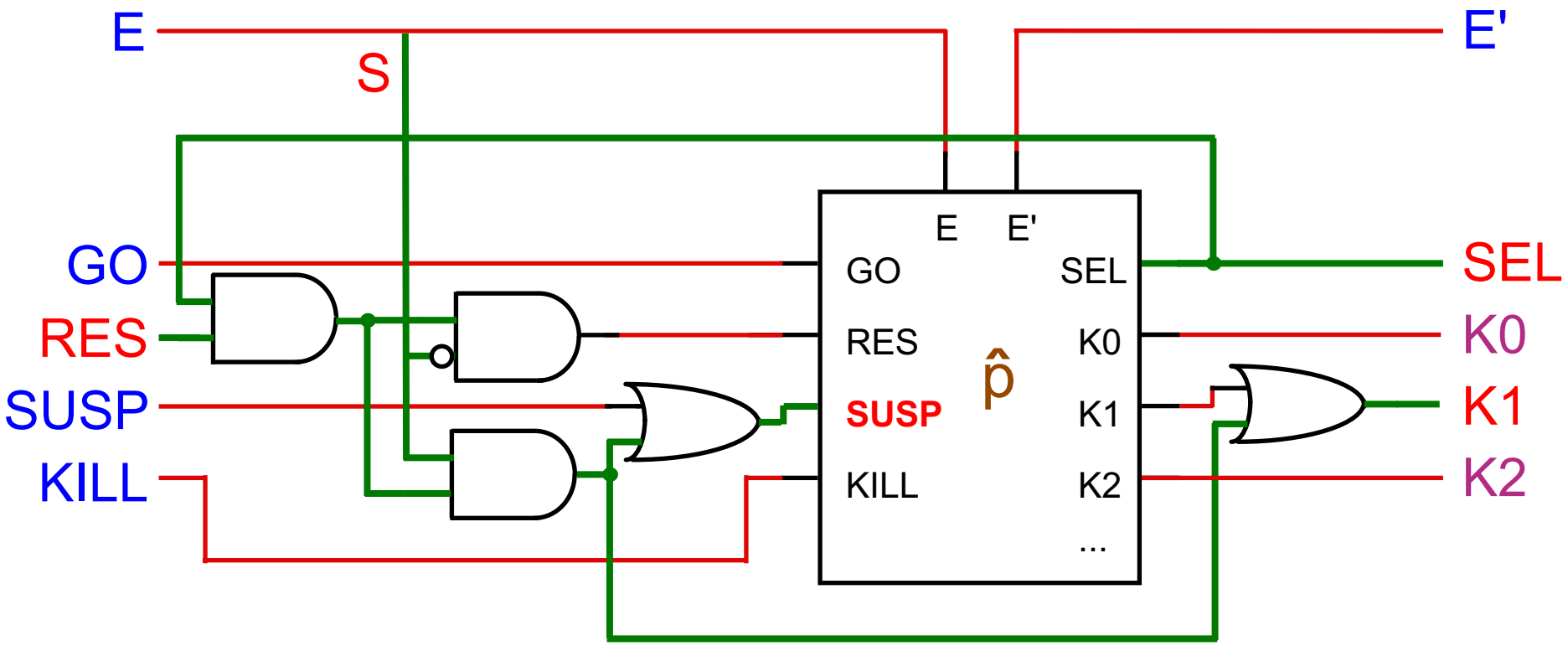
suspension retardée ⟹ GO est transmis à p

Circuit de « suspend p when S », S absent



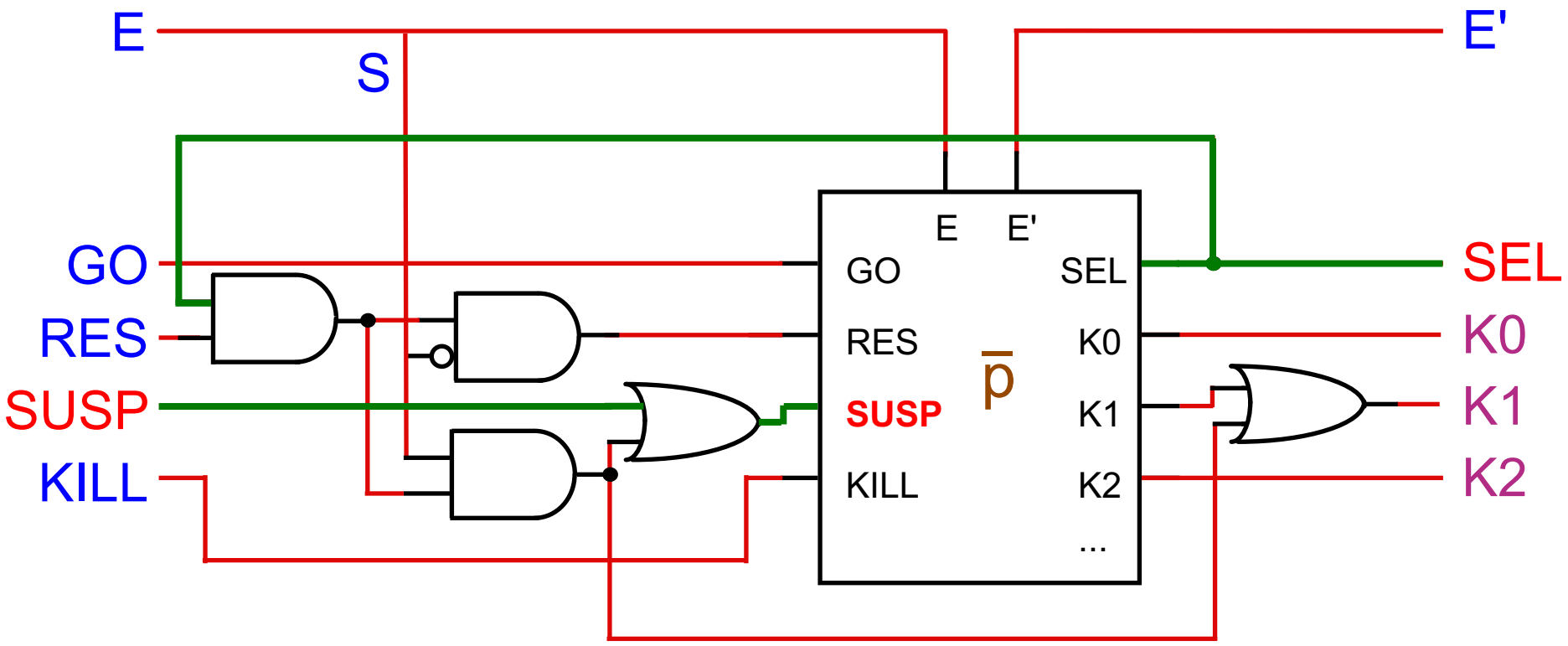
RES est transmis à p̂

Circuit de « suspend p when S », S présent



SUSP est transmis à p̂

« suspend p when S », SUSP propagé



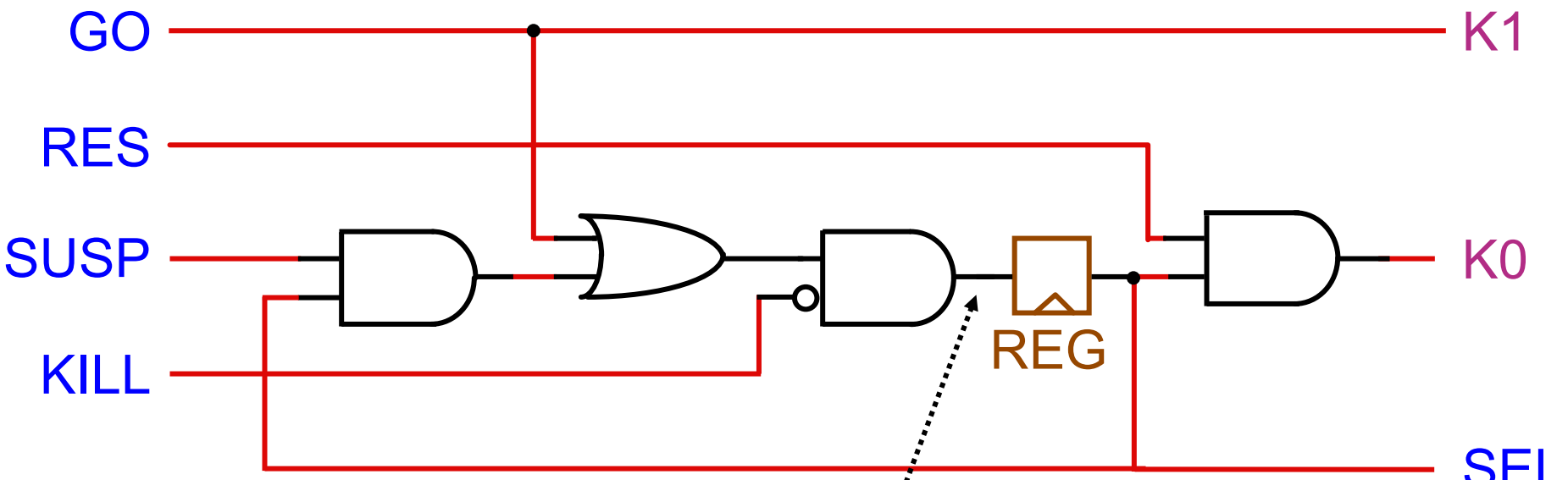
Si ne suspension englobante a mis l'entrée SUSP à 1, ce signal est transmis à $\bar{p}$.

Marche ausi pour p mort, SUSP laissant les registres à 0

Circuit de pause, final

1995:

suspend : être suspendu si activé. Fil : entrée SUSP



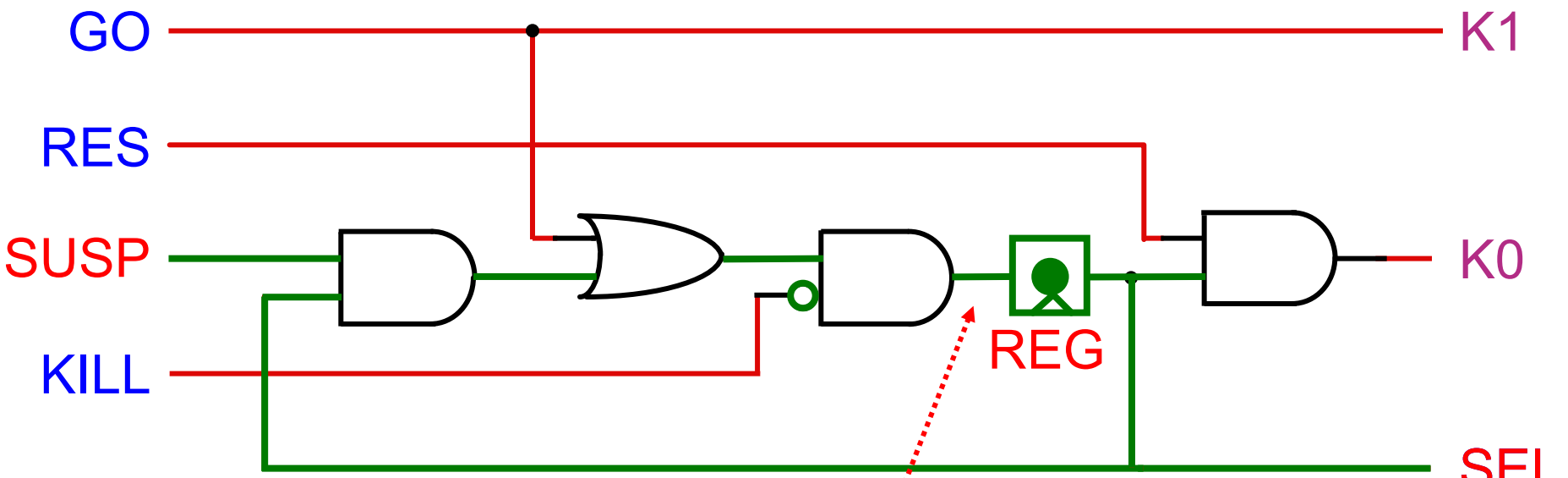
K0 = RES and REG

K1 = GO

SEL = REG

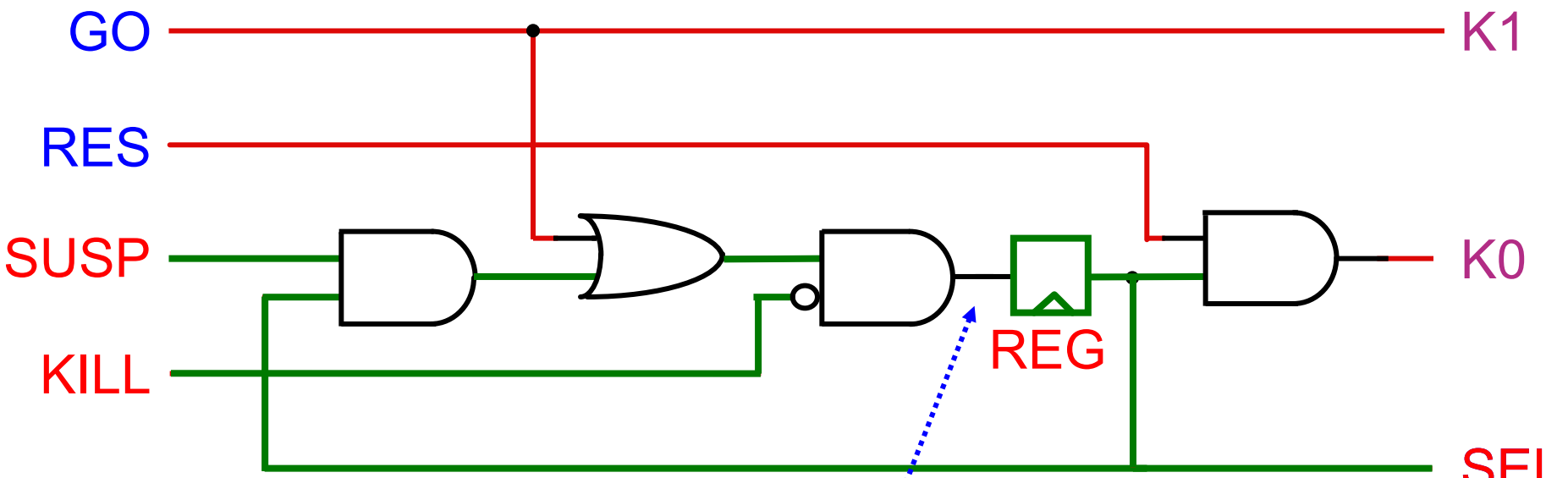
REG = pre((GO or (SUSP and SEL)) and not KILL)

Suspension d'une pause active



K0 = RES and REG
K1 = GO
SEL = REG
REG = pre((GO or (SUSP and SEL)) and not KILL)

Suspension tuée



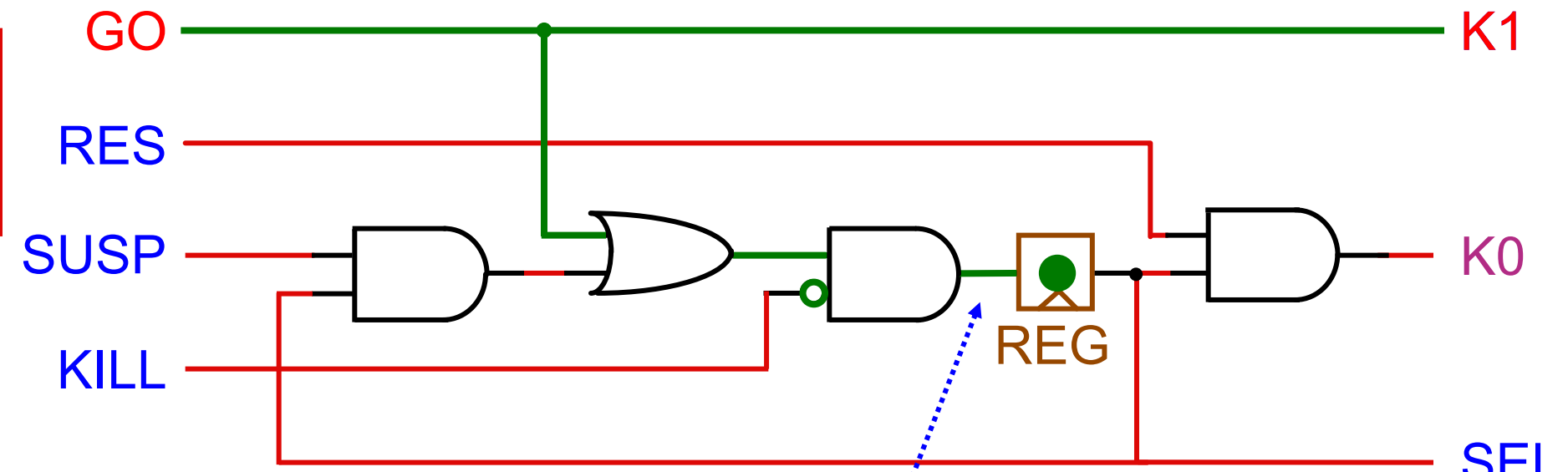
K0 = RES and REG

K1 = GO

SEL = REG

REG = pre((GO or (SUSP and SEL)) and not KILL)

Démarrage d'une pause



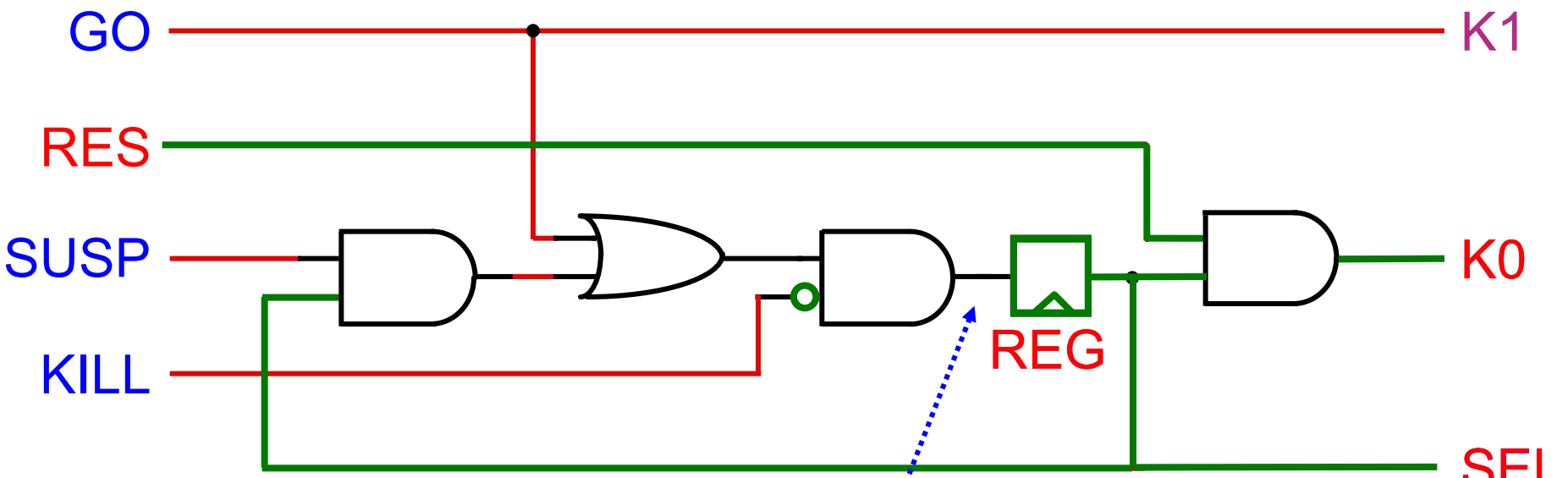
K0 = RES and REG

K1 = GO

SEL = REG

REG = pre((GO or (SUSP and SEL)) and not KILL)

Terminaison d'une pause active



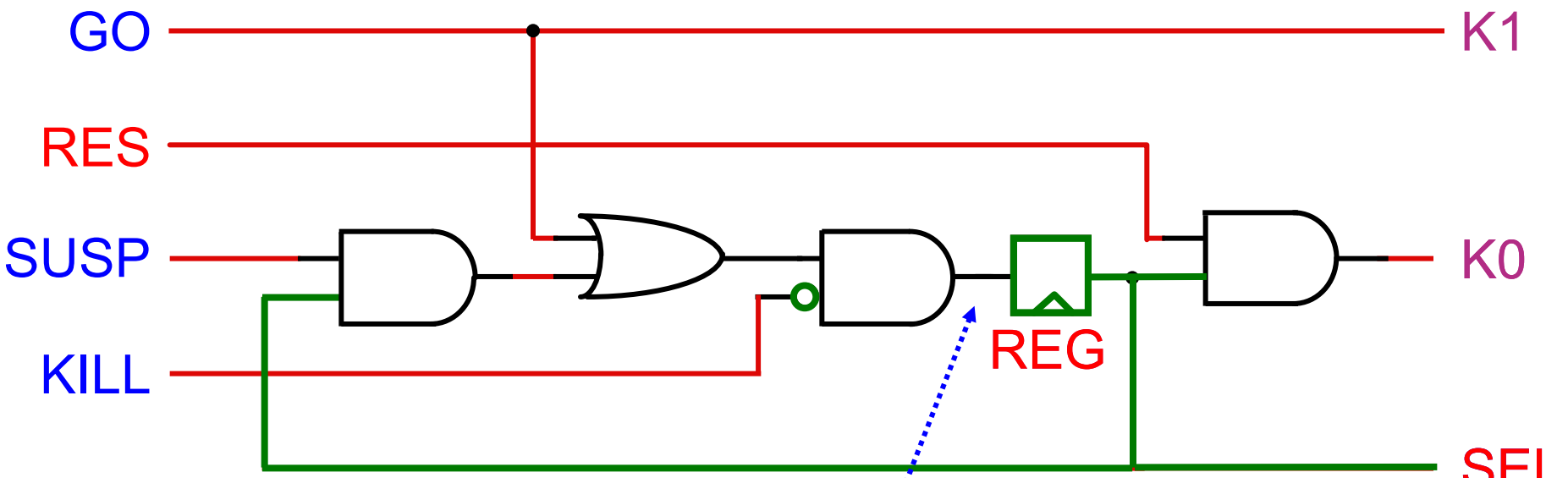
K0 = RES and REG

K1 = GO

SEL = REG

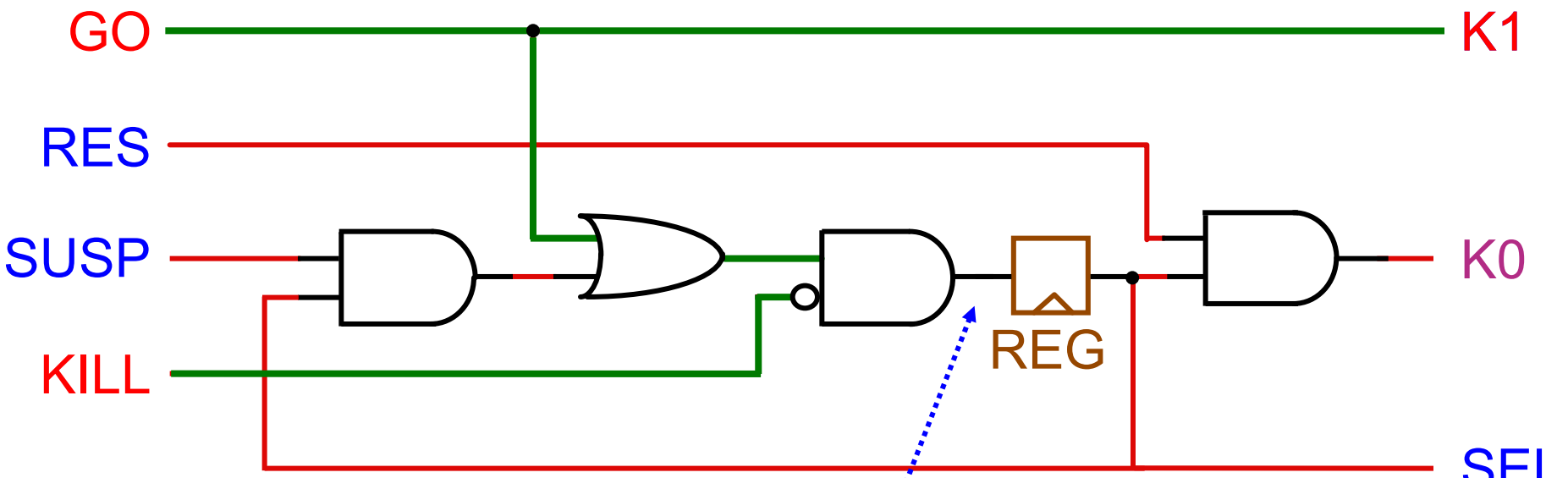
REG = pre((GO or (SUSP and SEL)) and not KILL)

Abort d'une pause active



K0 = RES and REG
K1 = GO
SEL = REG
REG = pre((GO or (SUSP and SEL)) and not KILL)

Pause mort-née



K0 = RES and REG
K1 = GO
SEL = REG
REG = pre((GO or (SUSP and SEL)) and not KILL)

Plan du cours

1. Rappels sur les circuits synchrones
2. La sémantique par termes marqués (états)
3. Interface des circuits
4. Circuits de pause, abort, trap et supend
5. Circuits de séquence et if

Règles de la séquence « p ; q »

$$\frac{\overline{p} \xrightarrow[I]{O,0} p \qquad q \xrightarrow[I]{O',k} \overline{q}'}{\overline{p};q \xrightarrow[I]{O\cup O',k} p;\overline{q}'}$$

(go-res-l-seq-0)

$$\frac{\overline{p} \xrightarrow[I]{O,k} \overline{p}' \qquad k\neq 0}{\overline{p};q \xrightarrow[I]{O,k} \overline{p}';q}$$

(go-res-l-seq-k>0)

$$\frac{\hat{q} \xrightarrow[I]{O,k} \overline{q}'}{p;\hat{q} \xrightarrow[I]{O,k} p;\overline{q}'}$$

(res-r-seq)

La séquence : séparation des cas pour p

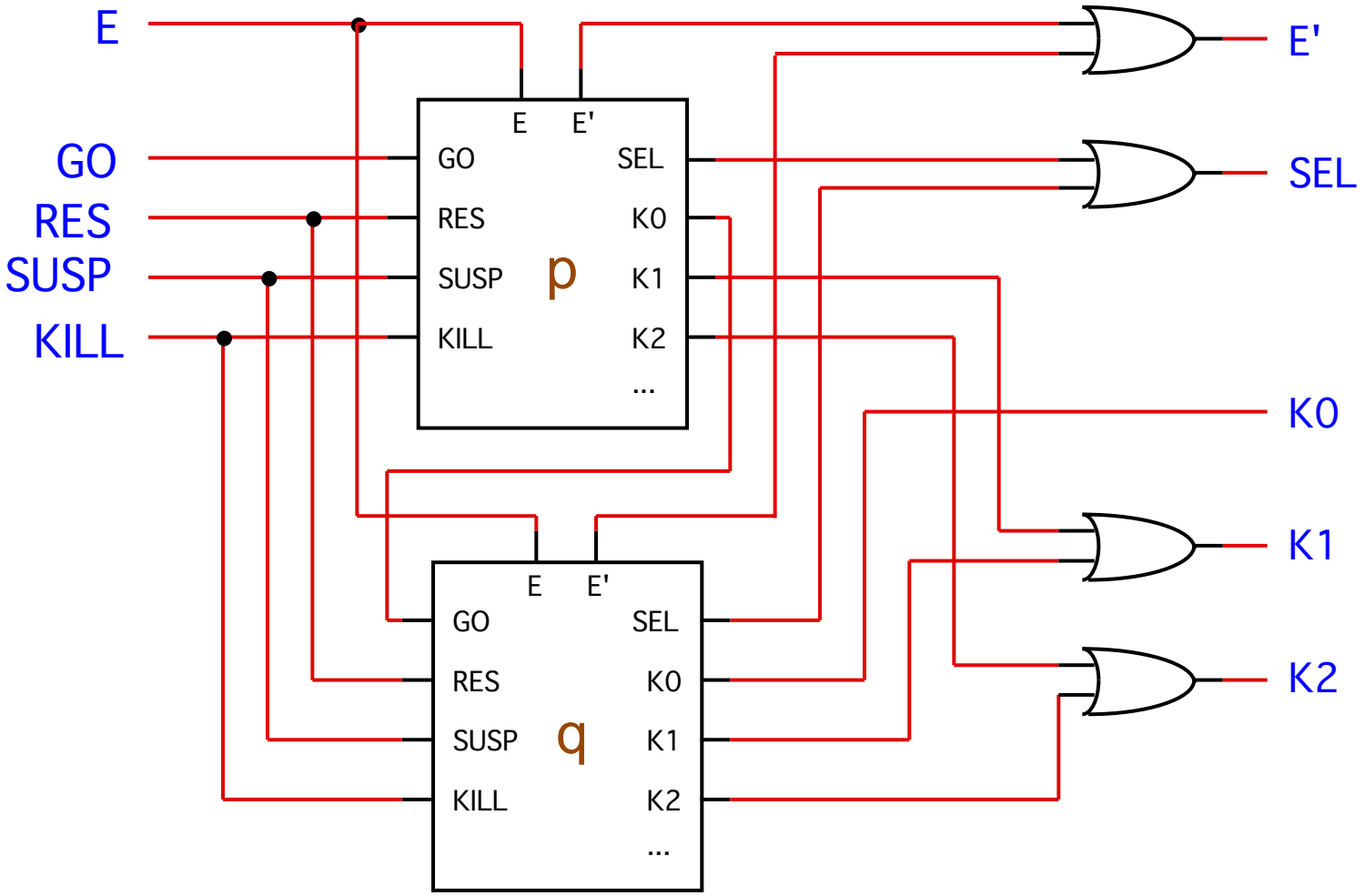
$$\frac{p \xrightarrow[I]{O,k} \overline{p}' \qquad k \neq 0}{p\,;q \xrightarrow[I]{O,k} \overline{p}'\,;q} \qquad (go\text{-}l\text{-}seq\text{-}k{>}0)$$

$$\frac{\hat{p} \xrightarrow[I]{O,k} \overline{p}' \qquad k \neq 0}{\hat{p}\,;q \xrightarrow[I]{O,k} \overline{p}'\,;q} \qquad (res\text{-}l\text{-}seq\text{-}k{>}0)$$

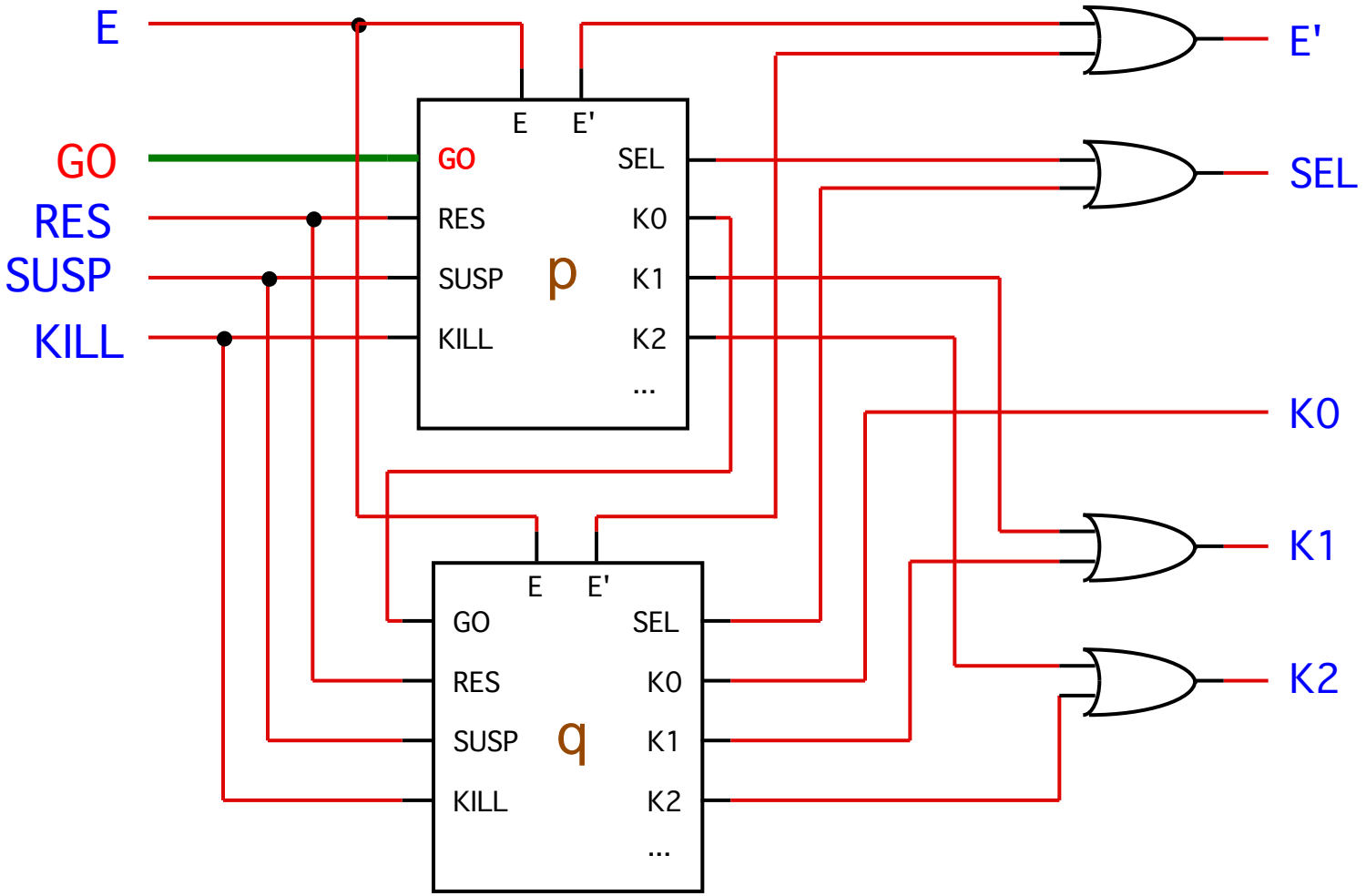
$$\frac{p \xrightarrow[I]{O,0} p \qquad q \xrightarrow[I]{O',k} \overline{q}'}{p\,;q \xrightarrow[I]{O\cup O',k} p\,;\overline{q}'} \qquad (go\text{-}seq\text{-}0)$$

$$\frac{\hat{p} \xrightarrow[I]{O,0} p \qquad q \xrightarrow[I]{O',k} \overline{q}'}{\hat{p}\,;q \xrightarrow[I]{O\cup O',k} p\,;\overline{q}'} \qquad (res\text{-}l\text{-}seq\text{-}0)$$

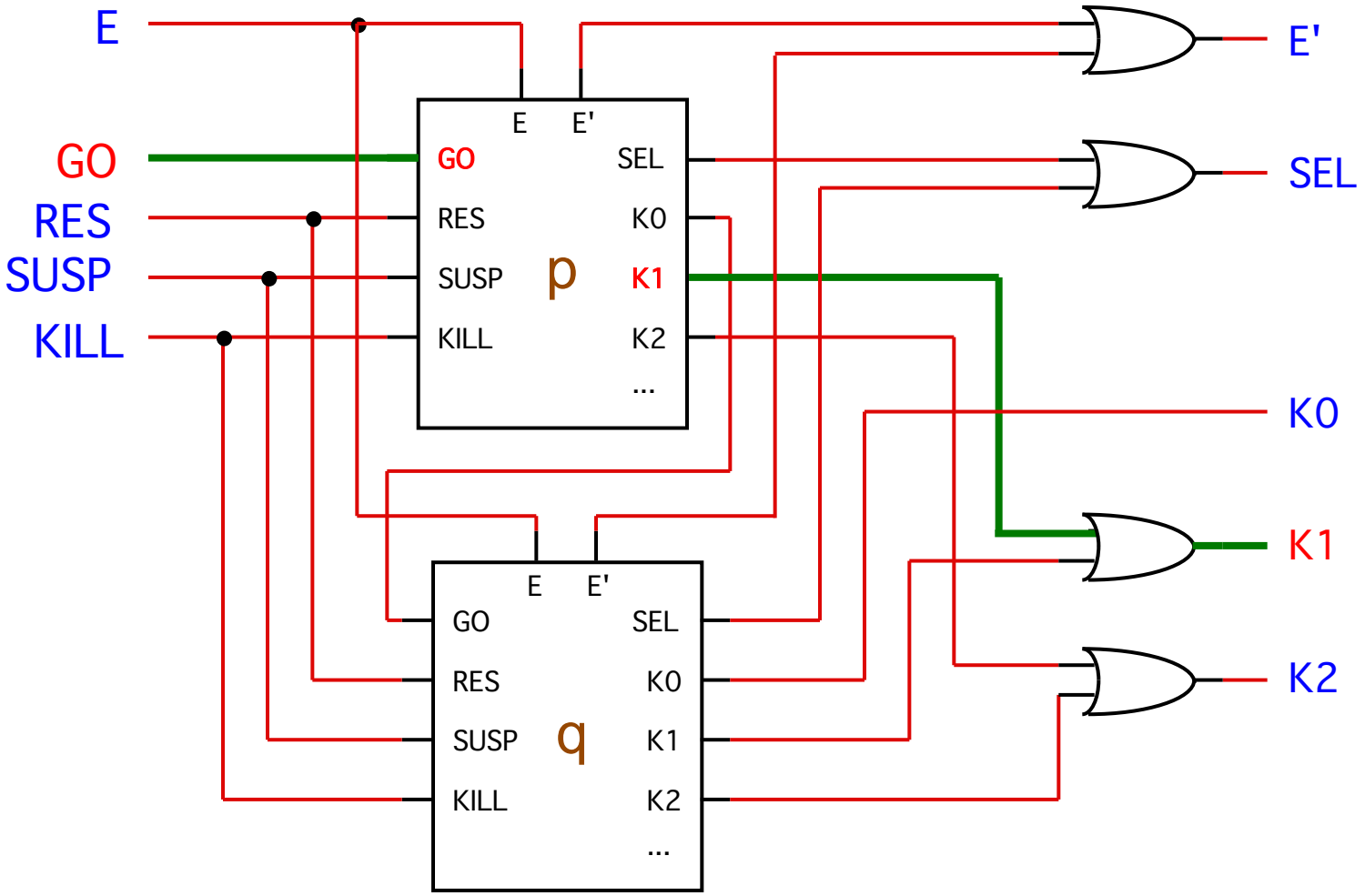
Circuit pour « p ; q »



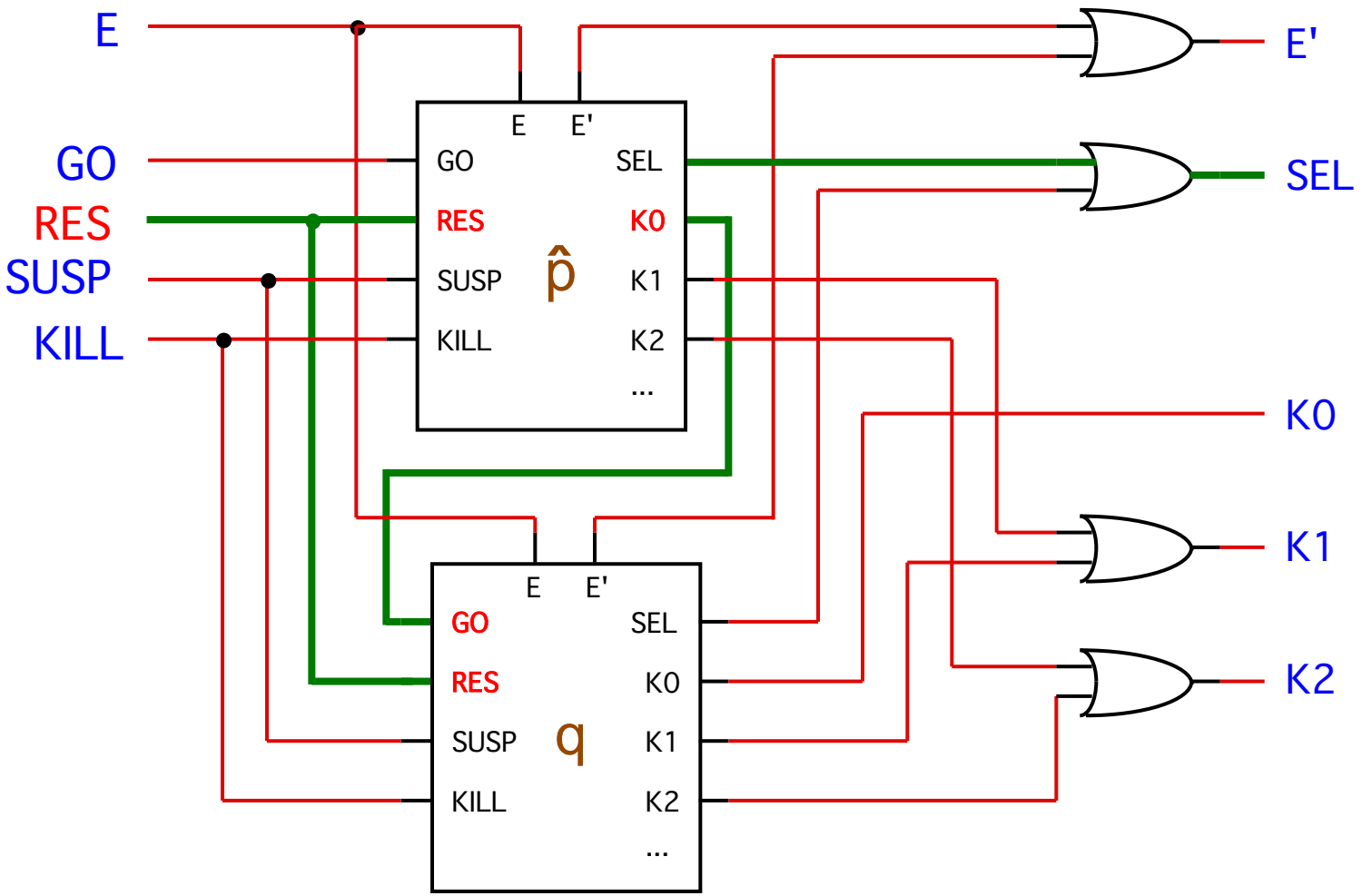
1. démarrage de p, qui pause



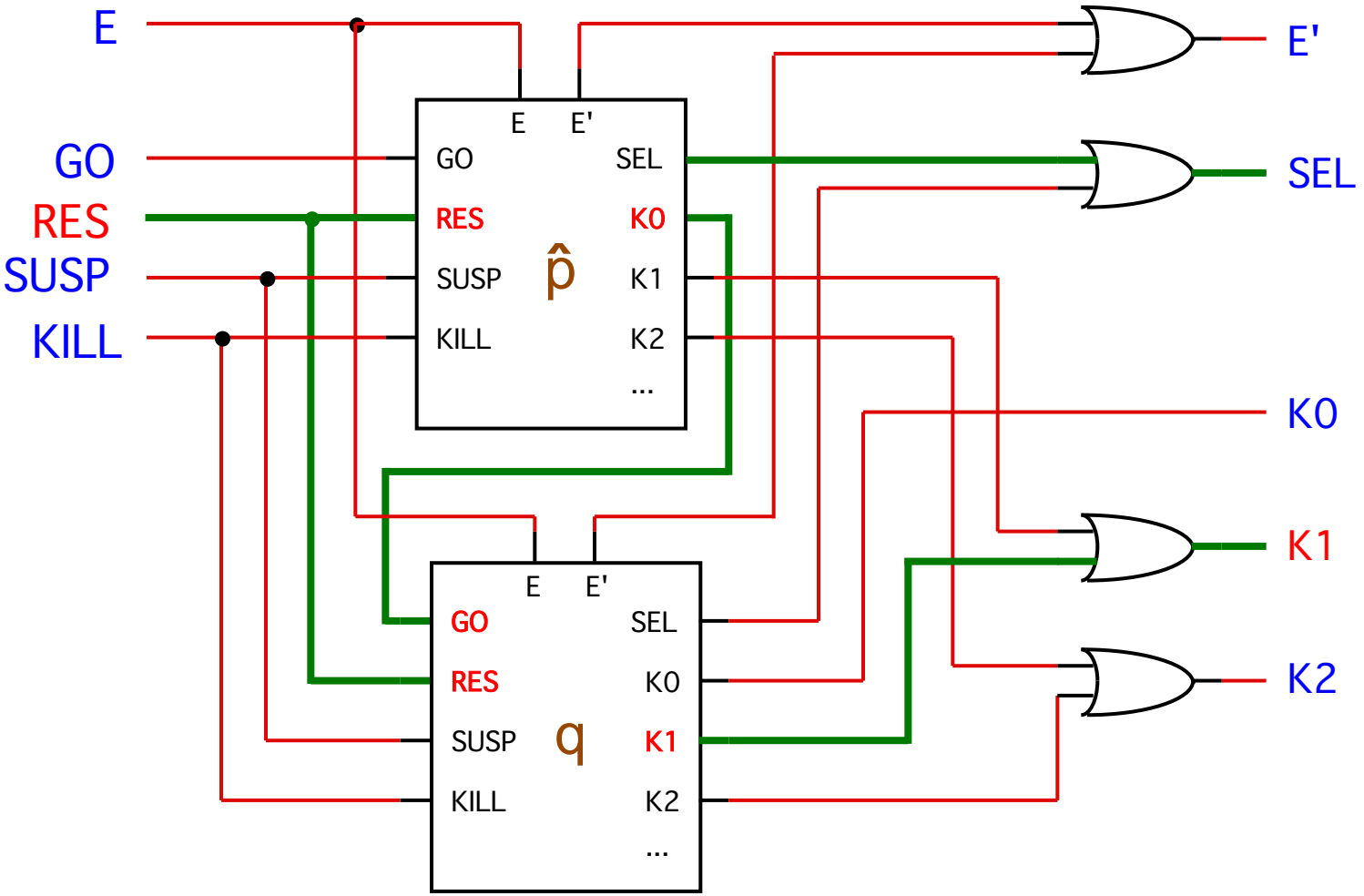
1. démarrage de p, qui pause



2. terminaison de p̂, démarrage de q qui pause

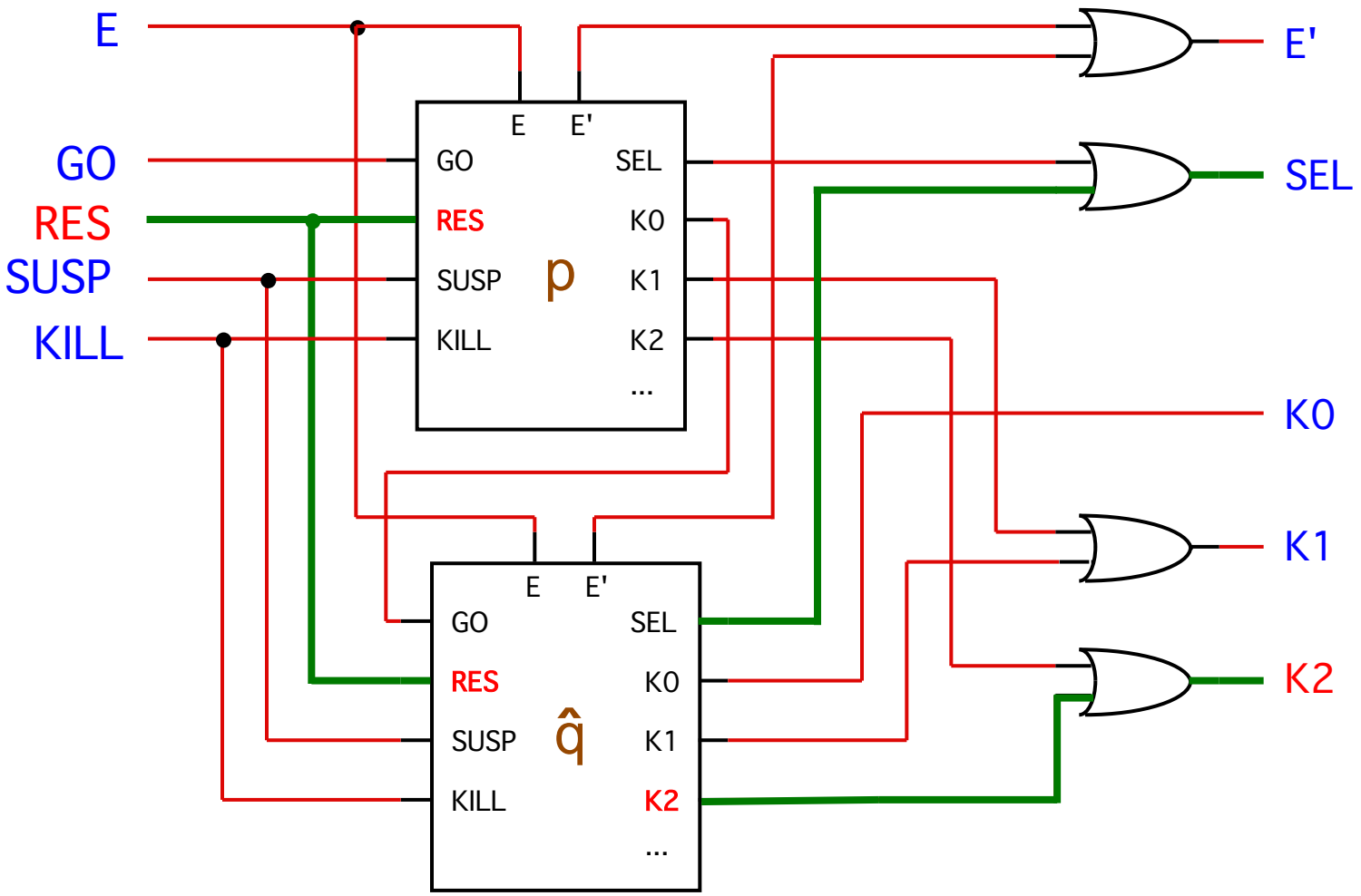


2. terminaison de p̂, démarrage de q qui pause



Note : RES est sans action sur q tant qu'il n'a pas démarré

3. fin de q̂ qui sort de la trappe 2



Note : RES est sans action sur p qui est mort

Invariant fondamental de la séquence

Pour tout état de toute séquence « p ; q »
p et q ne peuvent pas être tous deux sélectés :
SELp # SELq , *i.e.* not (SELp and SELq)

preuve :

1. L'état « p̂ ; q̂ » est inatteignable par les règles

Fondamental pour la vérification et l'optimisation

Règles de if

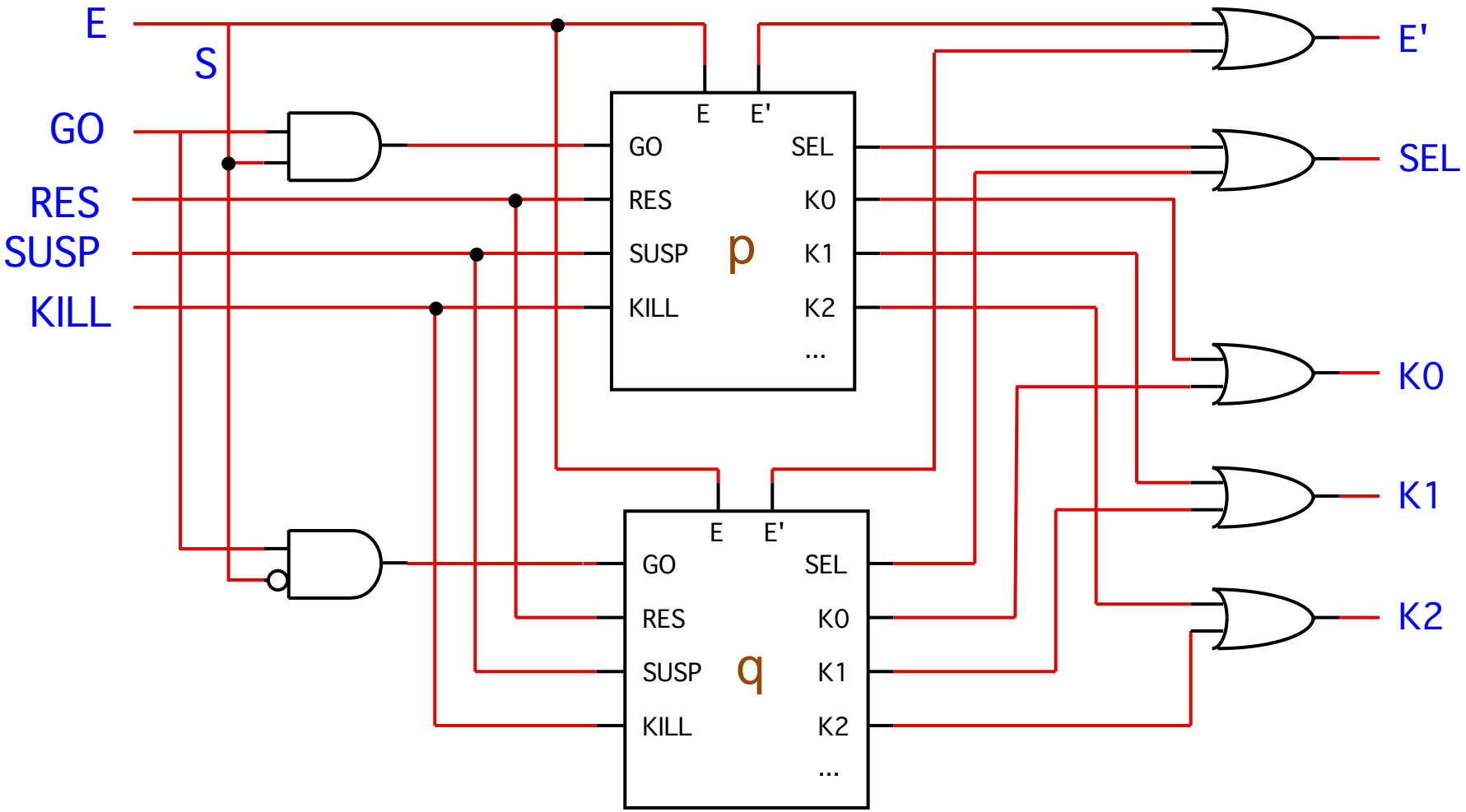
$$\frac{S \in I \qquad p \xrightarrow[I]{O,k} \overline{p}'}{\texttt{if } S \texttt{ then } p \texttt{ else } q \texttt{ end} \xrightarrow[I]{O,k} \texttt{if } S \texttt{ then } \overline{p}' \texttt{ else } q \texttt{ end}} \quad (go\text{-}present)$$

$$\frac{S \notin I \qquad q \xrightarrow[I]{O,k} \overline{q}'}{\texttt{if } S \texttt{ then } p \texttt{ else } q \texttt{ end} \xrightarrow[I]{O,k} \texttt{if } S \texttt{ then } p \texttt{ else } \overline{q}' \texttt{ end}} \quad (go\text{-}absent)$$

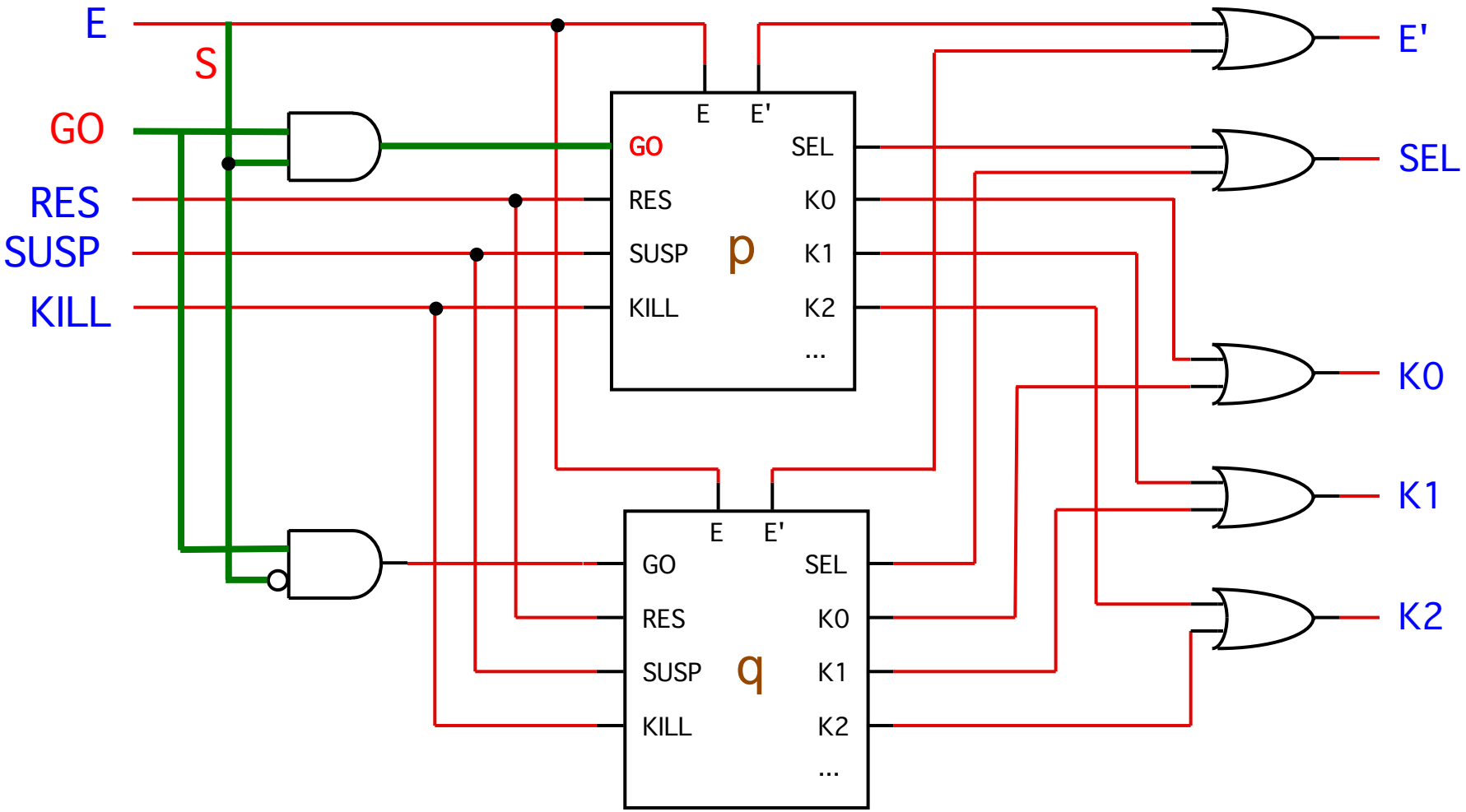
$$\frac{\hat{p} \xrightarrow[I]{O,k} \overline{p}'}{\texttt{if } S \texttt{ then } \hat{p} \texttt{ else } q \texttt{ end} \xrightarrow[I]{O,k} \texttt{if } S \texttt{ then } \overline{p}' \texttt{ else } q \texttt{ end}} \quad (res\text{-}then)$$

$$\frac{\hat{q} \xrightarrow[I]{O,k} \overline{q}'}{\texttt{if } S \texttt{ then } p \texttt{ else } \hat{q} \texttt{ end} \xrightarrow[I]{O,k} \texttt{if } S \texttt{ then } p \texttt{ else } \overline{q}' \texttt{ end}} \quad (res\text{-}else)$$

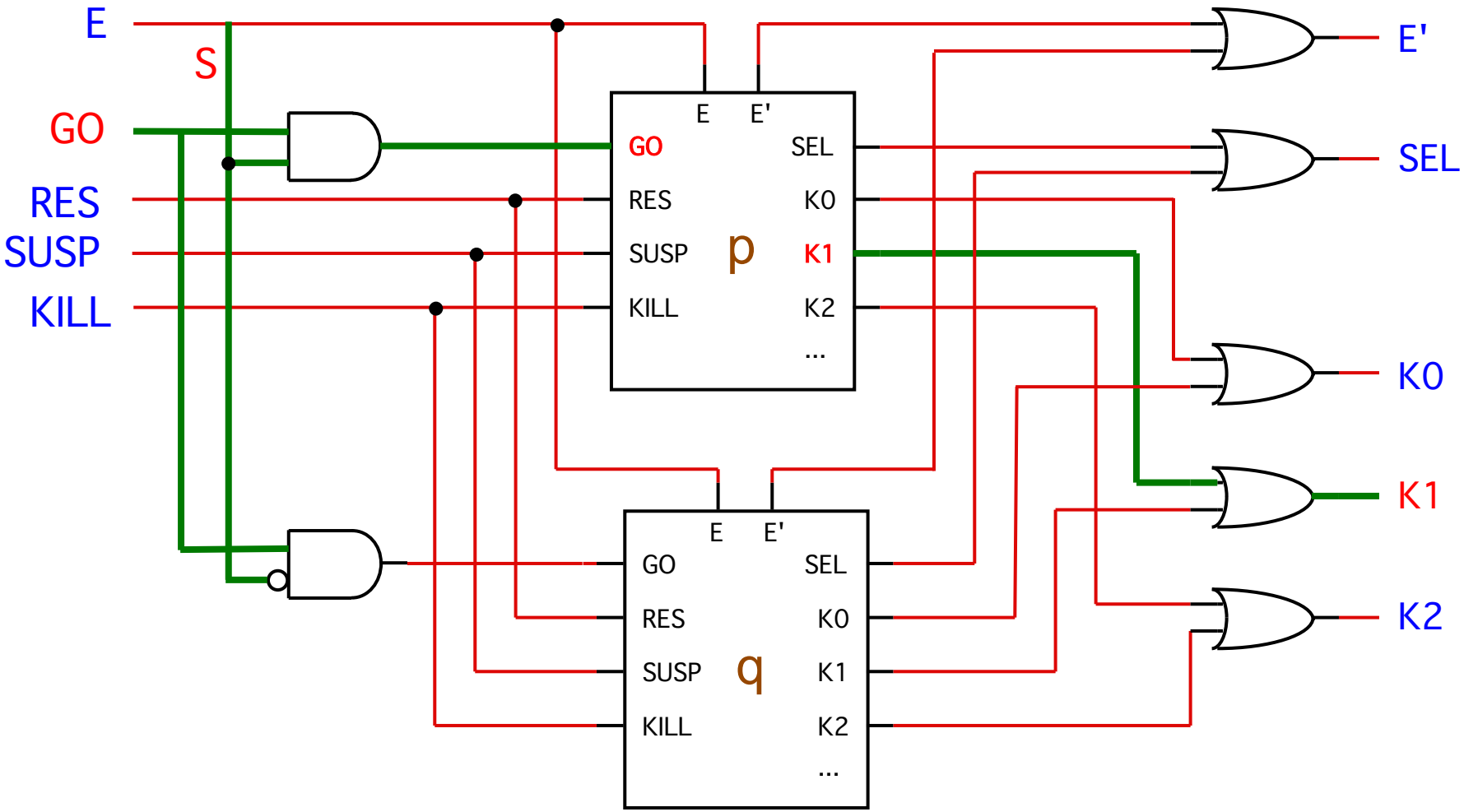
Circuit pour « if S then p else q end »



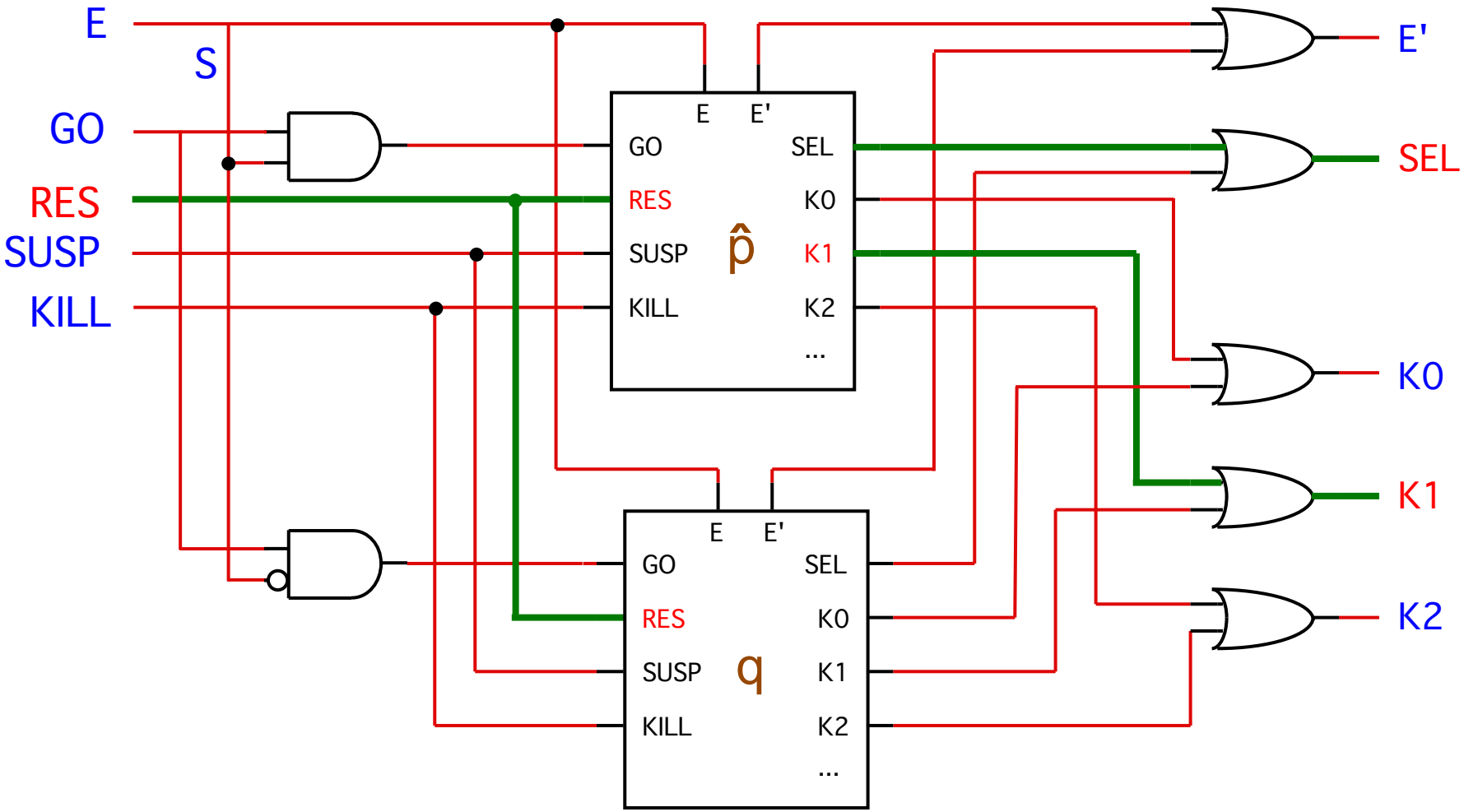
Circuit pour « if S then p else q end », S présent



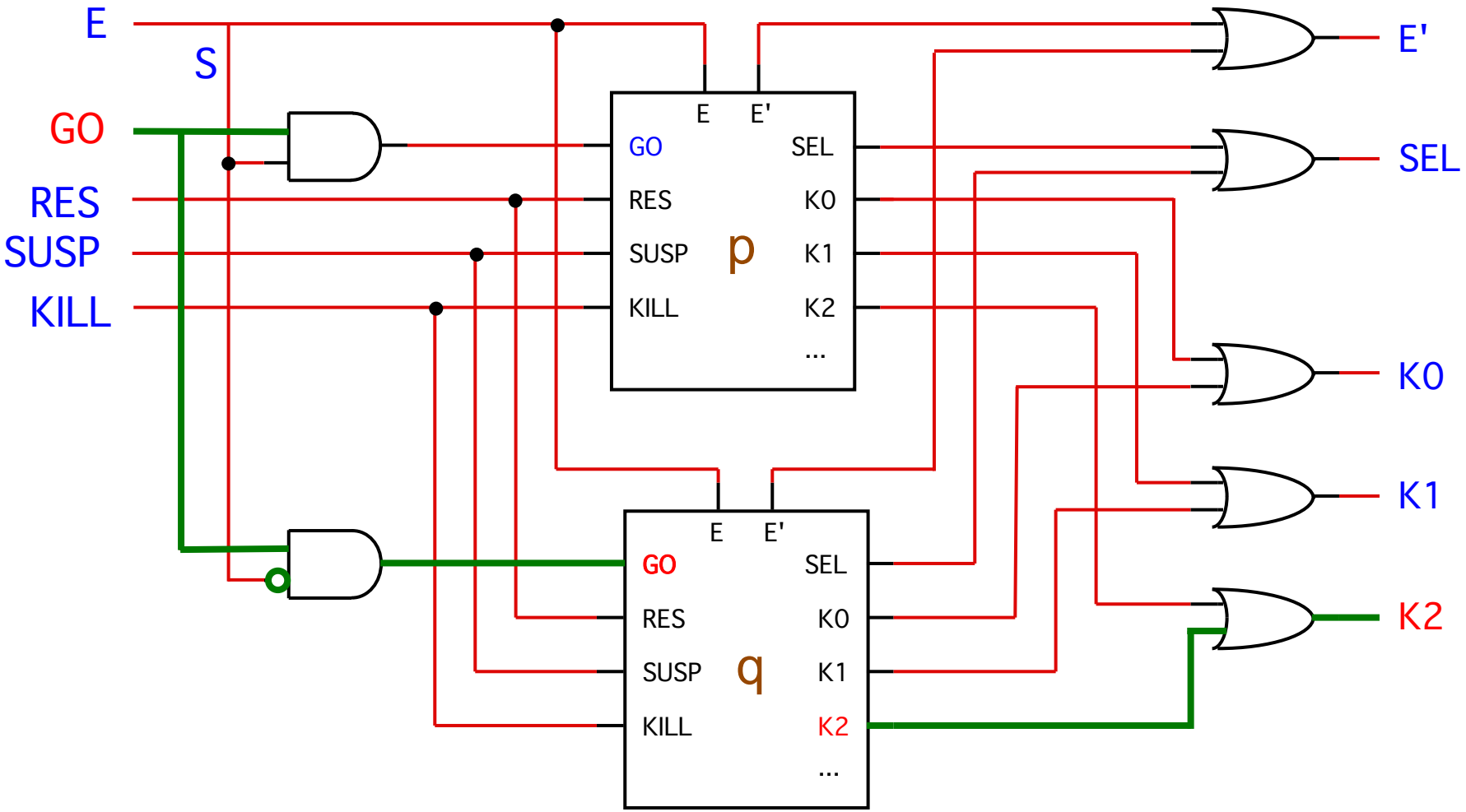
Circuit pour « if S then p else q end », S présent



« *if S then p else q end* », reprise de p̂



Circuit pour « if S then p else q end », S absent



Invariant fondamental du if

Pour tout état de tout test « if S then p else q end »
p et q ne peuvent pas être tous deux sélectés :
SELp # SELq, i.e. not (SELp and SELq)

preuve :

1. L'état « if S then p̂ else q̂ end » est inatteignable par les règles

Fondamental pour la vérification et l'optimisation

Plan du cours

1. Rappels sur les circuits synchrones
2. La sémantique par termes marqués (états)
3. Interface des circuits
4. Circuits de pause, abort, trap et supend
5. Circuits de séquence et if
6. Circuit du parallèle

Règles du parallèle

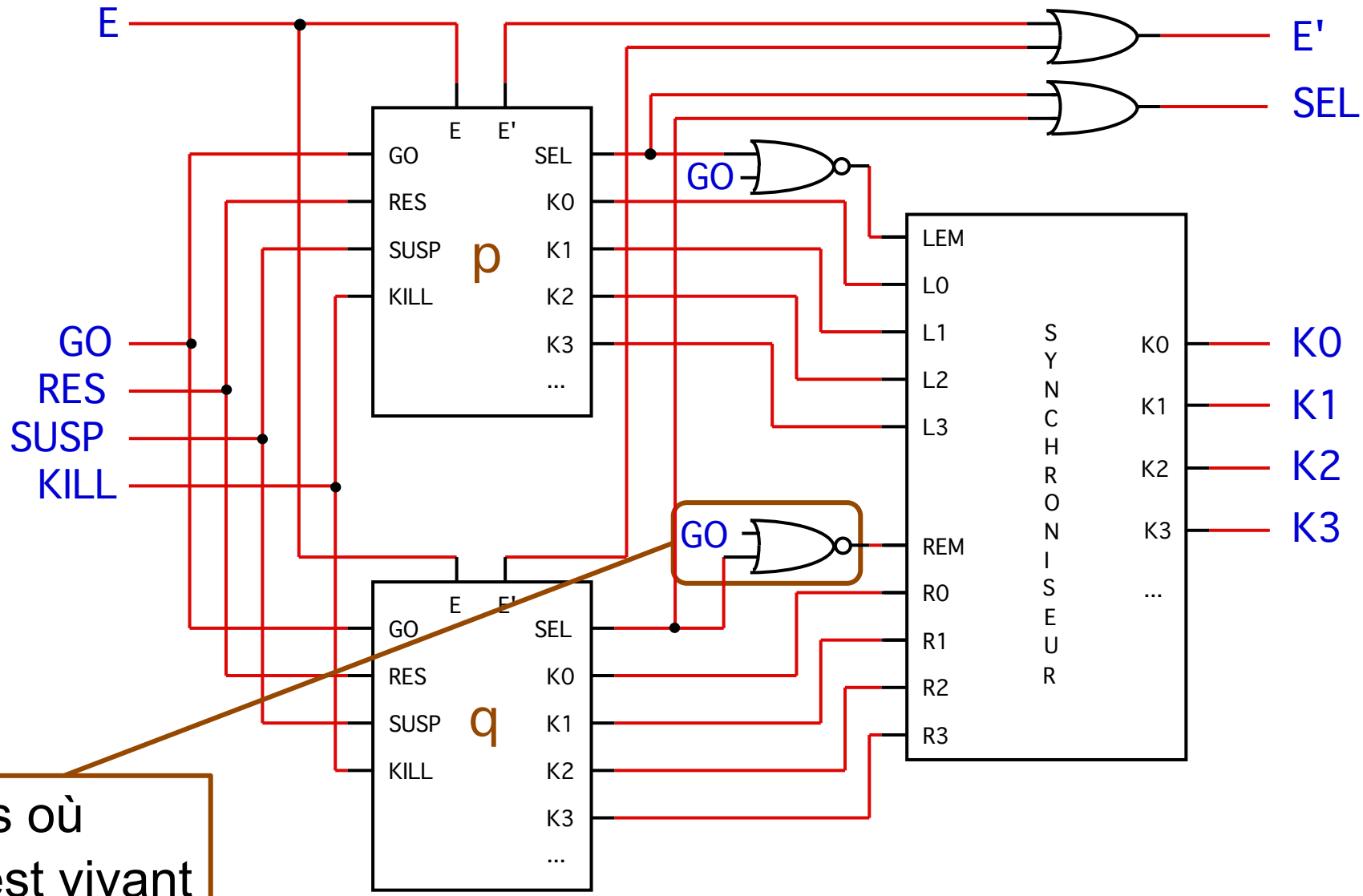
$$\frac{p \xrightarrow[I]{O,k} \overline{p}' \qquad q \xrightarrow[I]{O',l} \overline{q}' \qquad m{=}max(k,l)}{p \,||\, q \xrightarrow[I]{O \cup O',m} \overline{p}' \,||\, \overline{q}'} \quad (go\text{-}parallel)$$

$$\frac{\hat{p} \xrightarrow[I]{O,k} \overline{p}' \qquad \hat{q} \xrightarrow[I]{O',l} \overline{q}' \qquad m{=}max(k,l)}{\hat{p} \,||\, \hat{q} \xrightarrow[I]{O \cup O',m} \overline{p}' \,||\, \overline{q}'} \quad (res\text{-}lr\text{-}parallel)$$

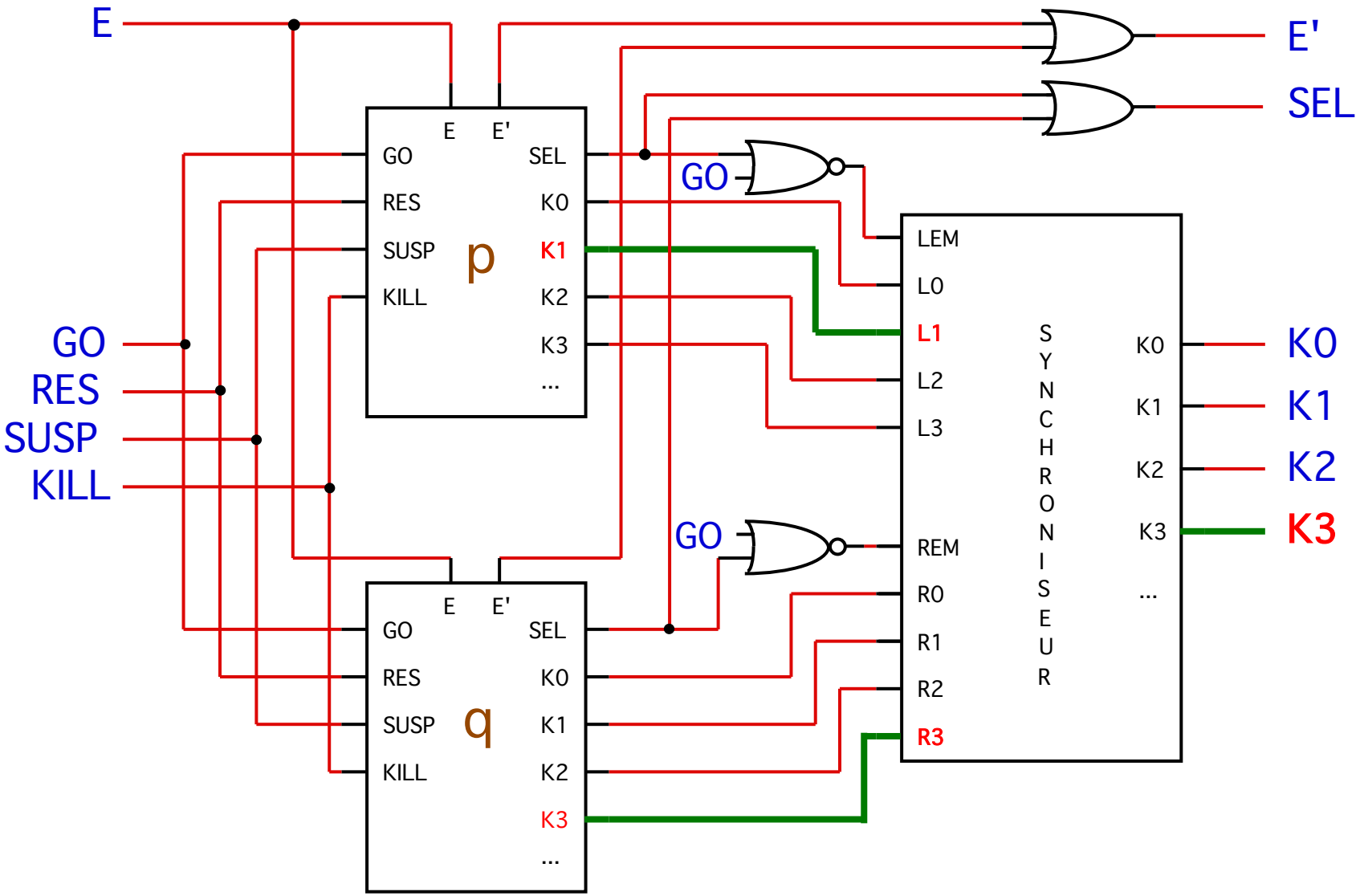
$$\frac{\hat{p} \xrightarrow[I]{O,k} \overline{p}'}{\hat{p} \,||\, q \xrightarrow[I]{O \cup O',k} \overline{p}' \,||\, q} \quad (res\text{-}l\text{-}parallel)$$

$$\frac{\hat{q} \xrightarrow[I]{O',l} \overline{q}'}{p \,||\, \hat{q} \xrightarrow[I]{O \cup O',l} p \,||\, \overline{q}'} \quad (res\text{-}r\text{-}parallel)$$

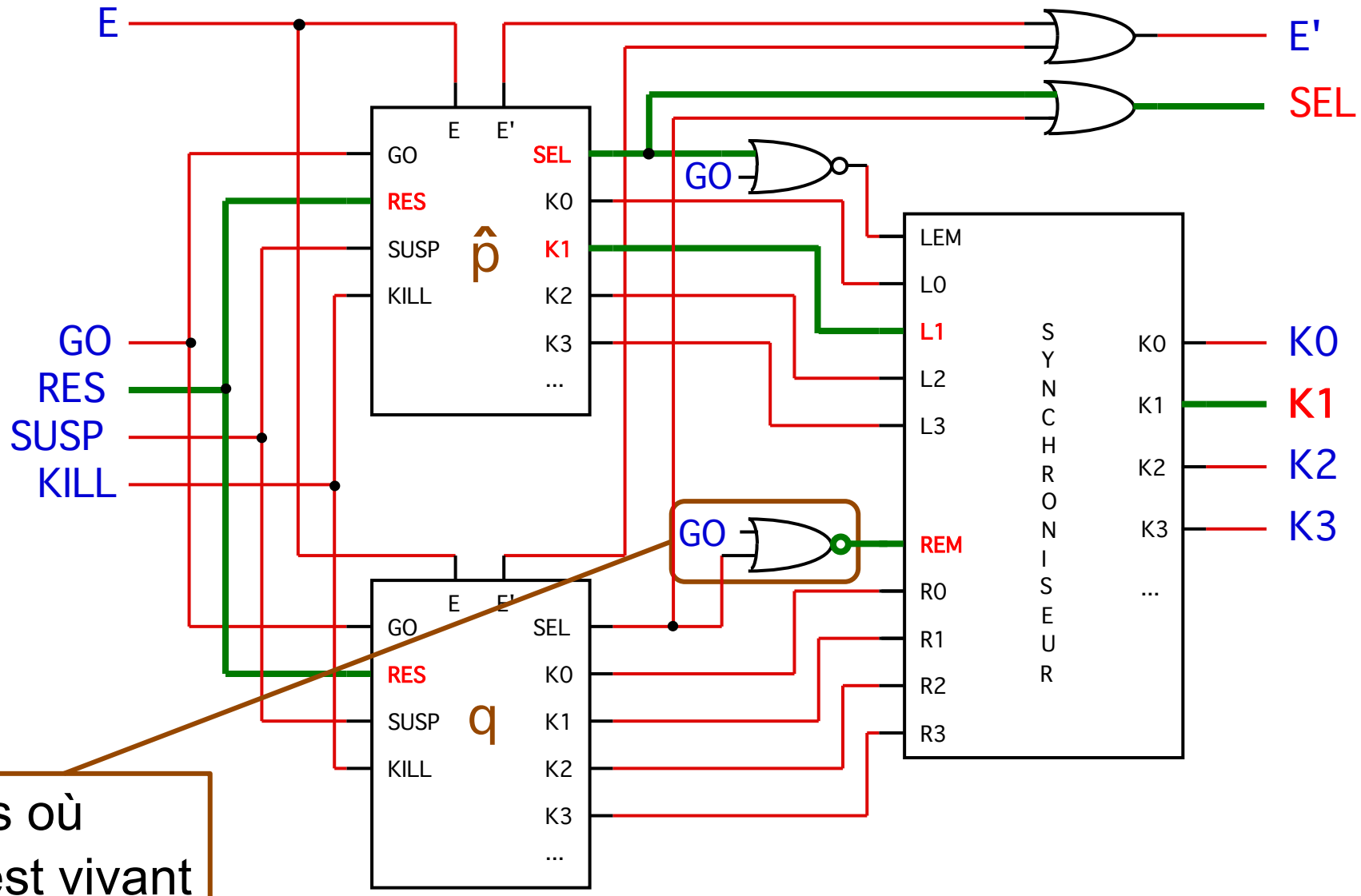
Circuit pour p || q



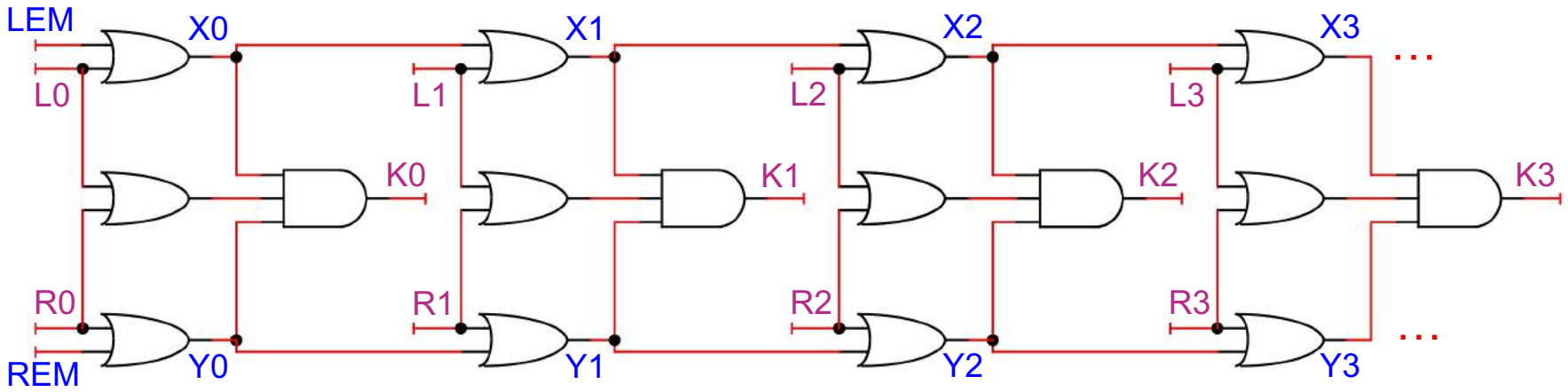
Circuit pour p || q, calcul du max



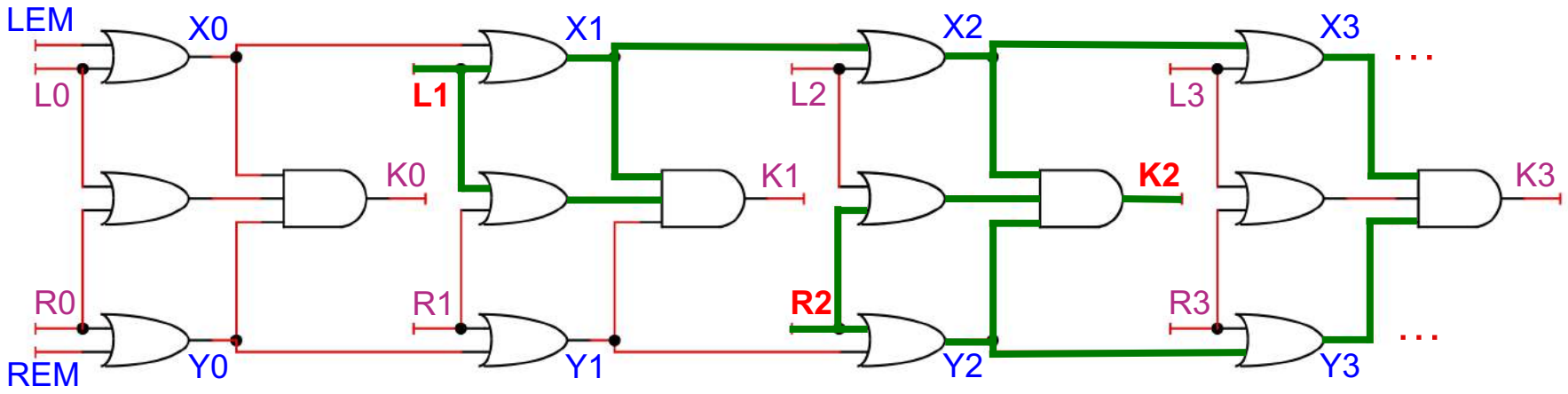
Circuit pour p̂ || q avec q mort



Le synchroniseur (max-unit 2-adique, Gonthier)



Le synchroniseur (max-unit 2-adique, Gonthier)

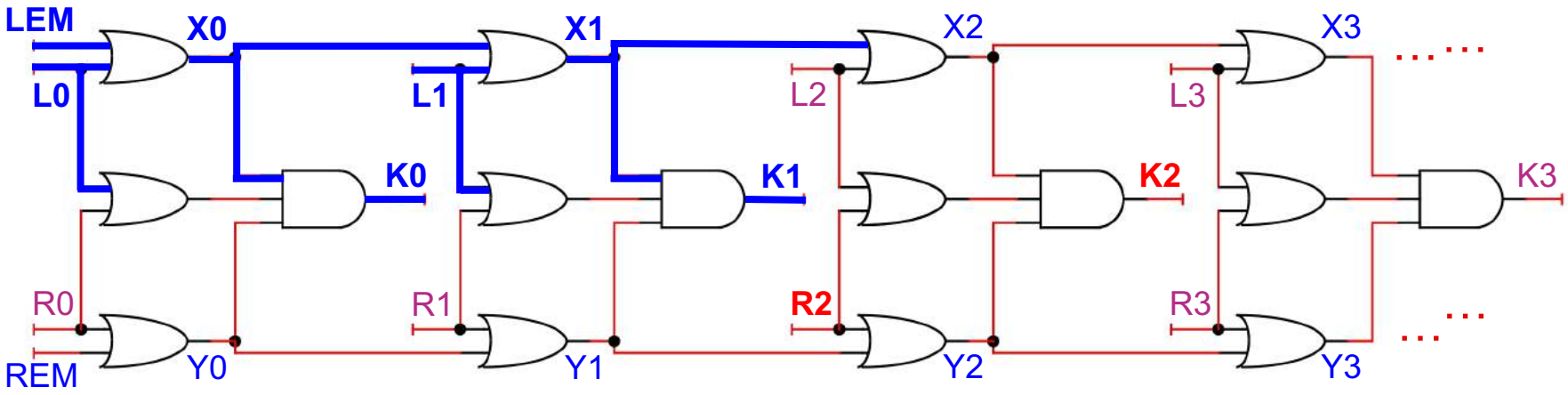


Calcul du max :

1. un seul des LEM / Lm (REM, Rn) vaut 1 à chaque instant
2. si Lm vaut 1, en identifiant LEM et L0, alors
   a. tous les Xi valent 0 pour $i < m$
   b. tous les Xj valent 1 pour $j > m$
3. Idem pour le REM / Rn valant 1
4. Donc, pour $k = \max(m,n)$, seul Kk vaut 1,

CQFD

Propriété essentielle : la propagation des 0

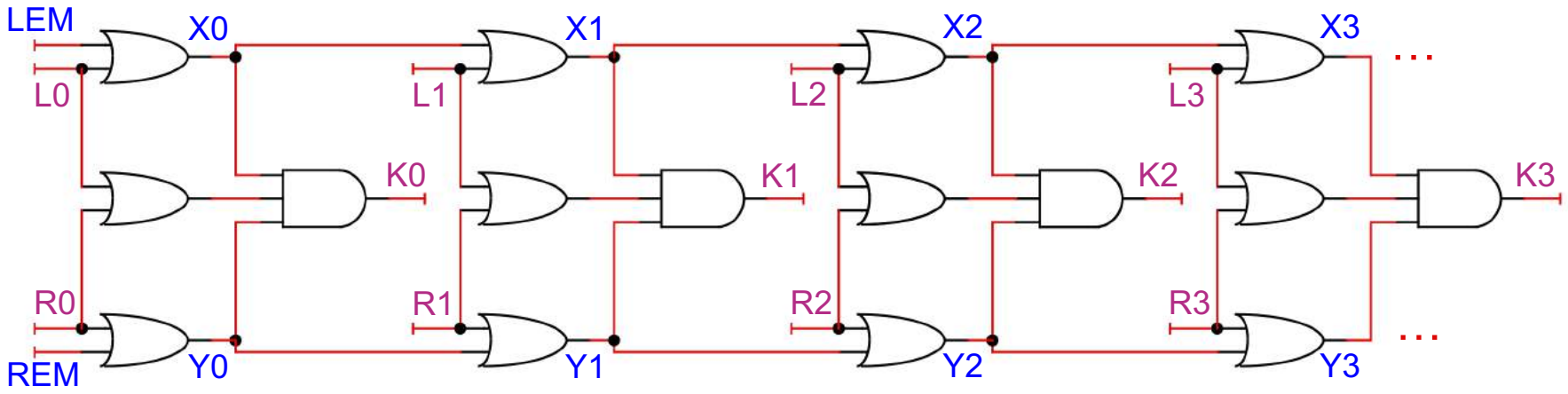


Dès qu'on sait LEM=L0=L1=0, on dérive K0=K1=0, même en ne sachant rien sur les Rj !

Propager les 1, c'est calculer *Must*

Propager les 0, c'est calculer *Cannot*

Le synchroniseur (max-unit 2-adique, Gonthier)



Marche même si L et R ont plusieurs 1 :
calcule en fait un Max d'ensembles d'entiers,

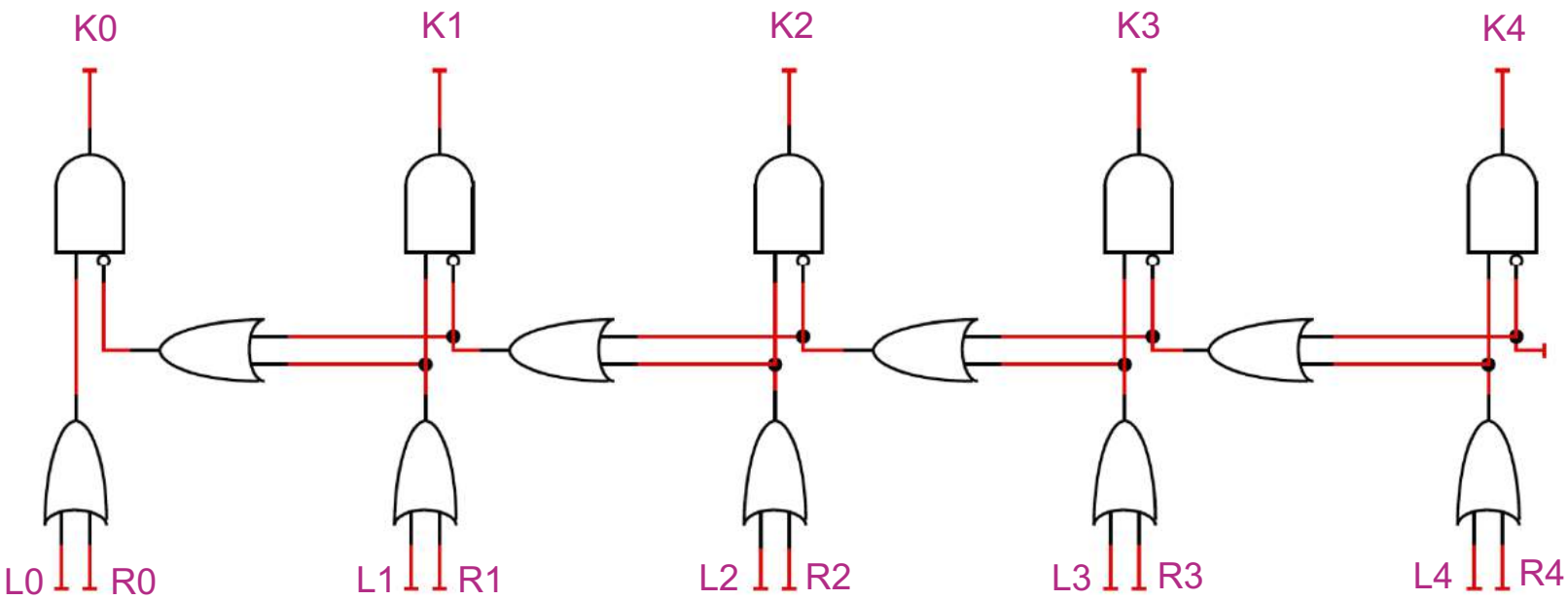
$$Max(L,R) = \{ max(l,r) \mid l \in L, r \in R \}$$

qui se code en 2-adiques par la formule magique de Gonthier

$$Max(L,R) = (L \text{ or } R) \text{ and } (L \text{ or } -L) \text{ and } (R \text{ or } -L)$$

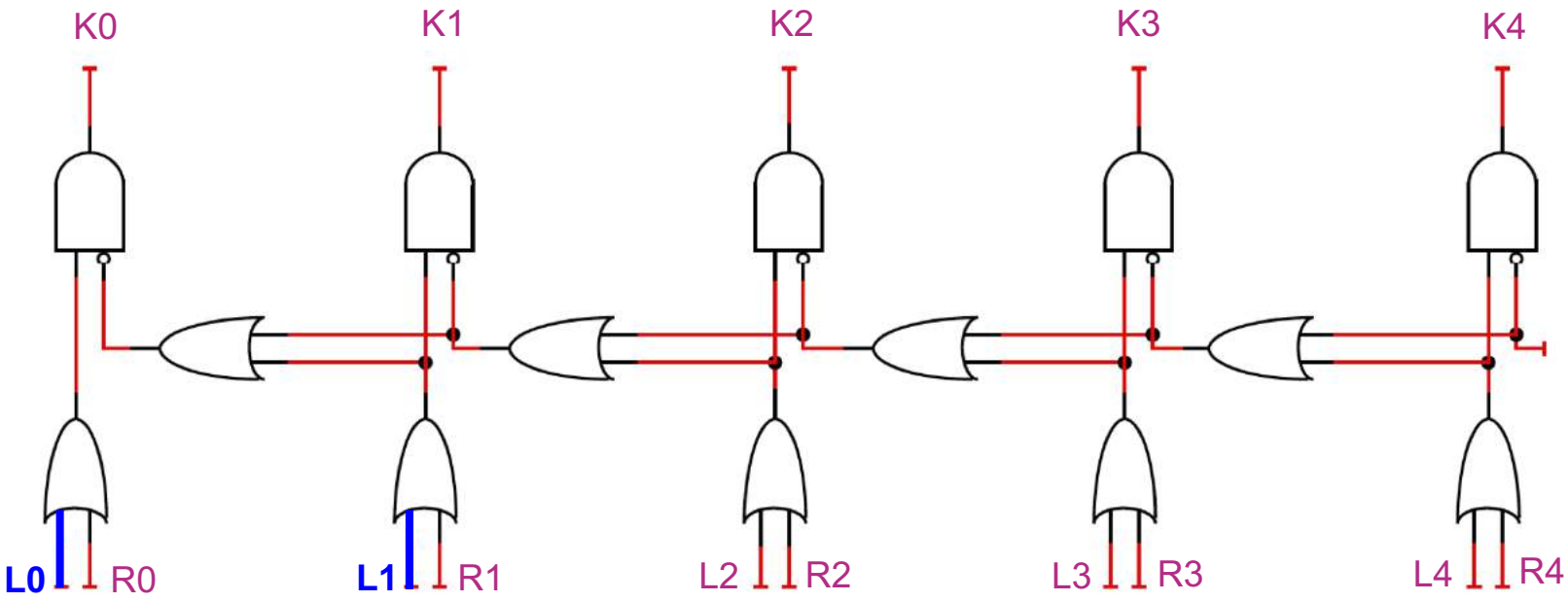
(voir cours SMT du 23/03/2016 pour la preuve automatique en Why)

La Daisy Chain plus classique pour max



La Daisy Chain plus classique pour max

ne propage pas les 0 !



Plan du cours

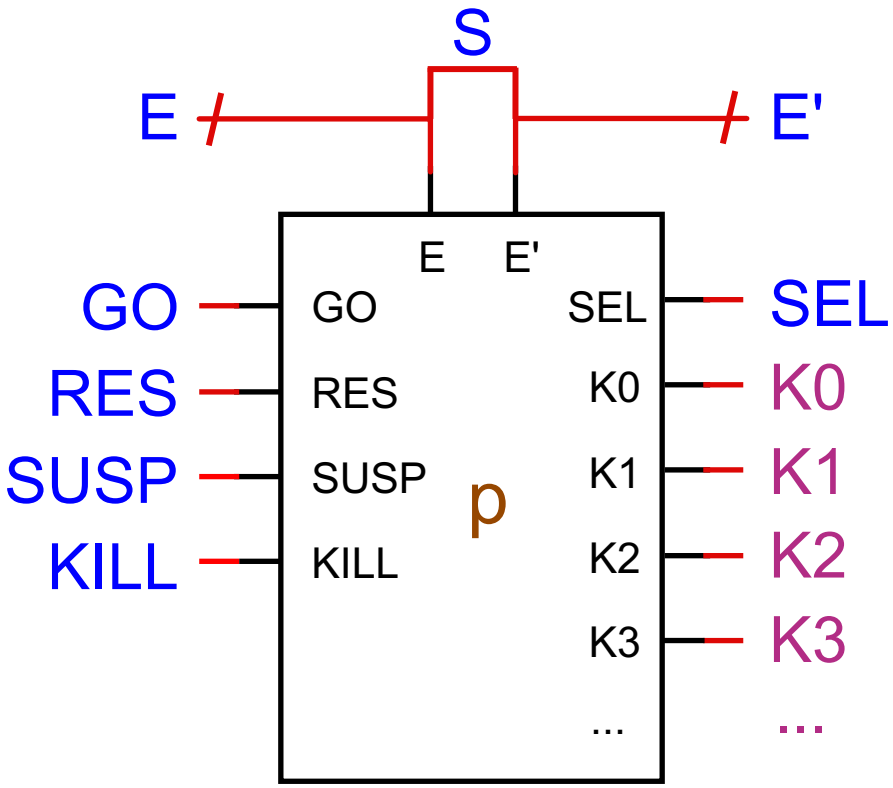
1. Rappels sur les circuits synchrones
2. La sémantique par termes marqués (états)
3. Interface des circuits
4. Circuits de pause, abort, trap et supend
5. Circuits de séquence, et if
6. Circuit de parallèle
7. Circuit de signal

Règles comportementales de « signal S in p end »

$$\frac{\overline{p} \xrightarrow[I \cup \{S\}]{O,k} \overline{p}' \qquad S \in O}{\texttt{signal } S \texttt{ in } \overline{p} \texttt{ end} \xrightarrow[I]{O \setminus \{S\},k} \texttt{signal } S \texttt{ in } \overline{p}' \texttt{ end}} \quad (go\text{-}res\text{-}signal+)$$

$$\frac{\overline{p} \xrightarrow[I \setminus \{S\}]{O,k} \overline{p}' \qquad S \notin O}{\texttt{signal } S \texttt{ in } p \texttt{ end} \xrightarrow[I]{O,k} \texttt{signal } S \texttt{ in } p' \texttt{ end}} \quad (go\text{-}res\text{-}signal-)$$

Circuit de « signal S in p end »

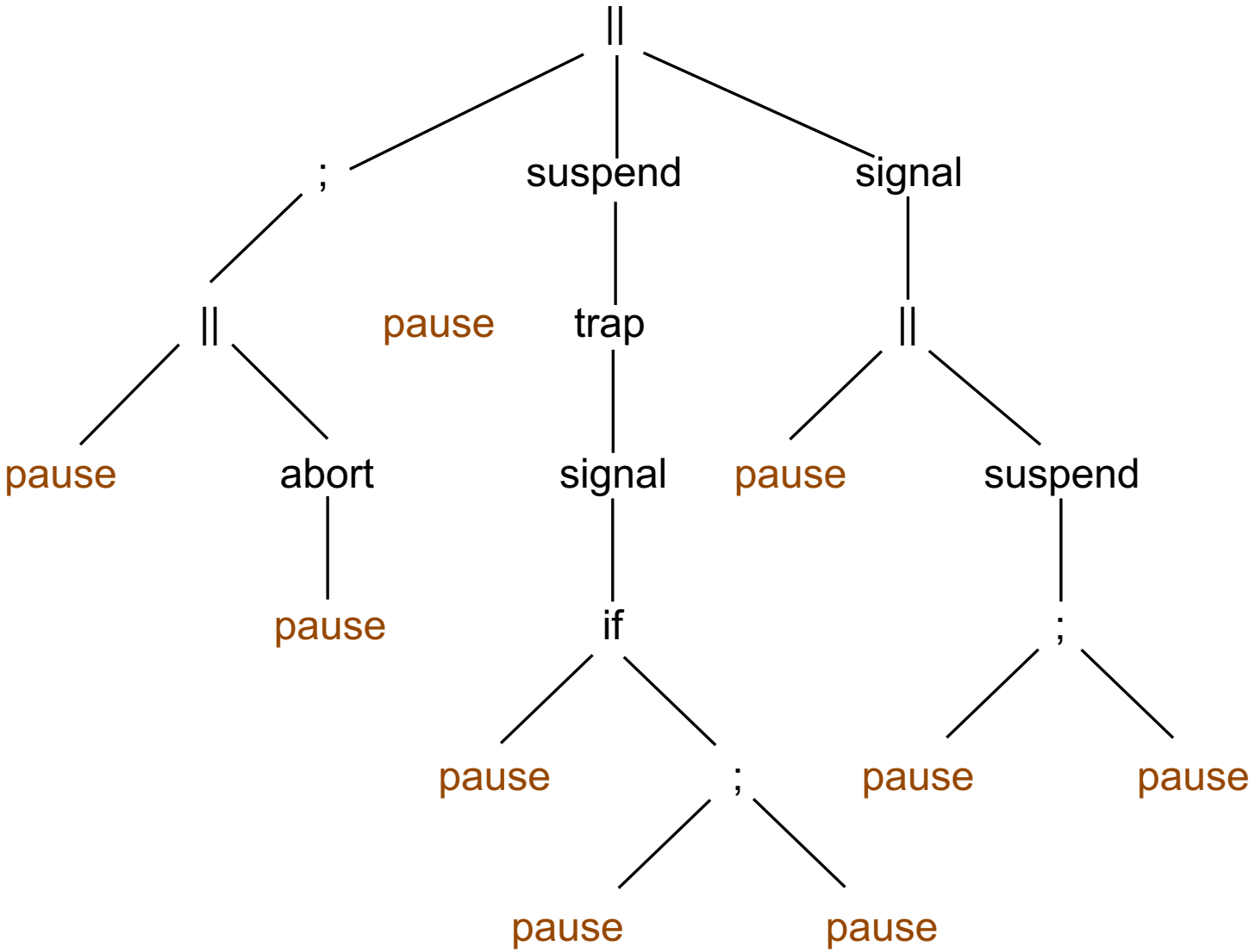


Peut engendrer des cycles, mais
programme constructif ⟹ cycles électriquement sains

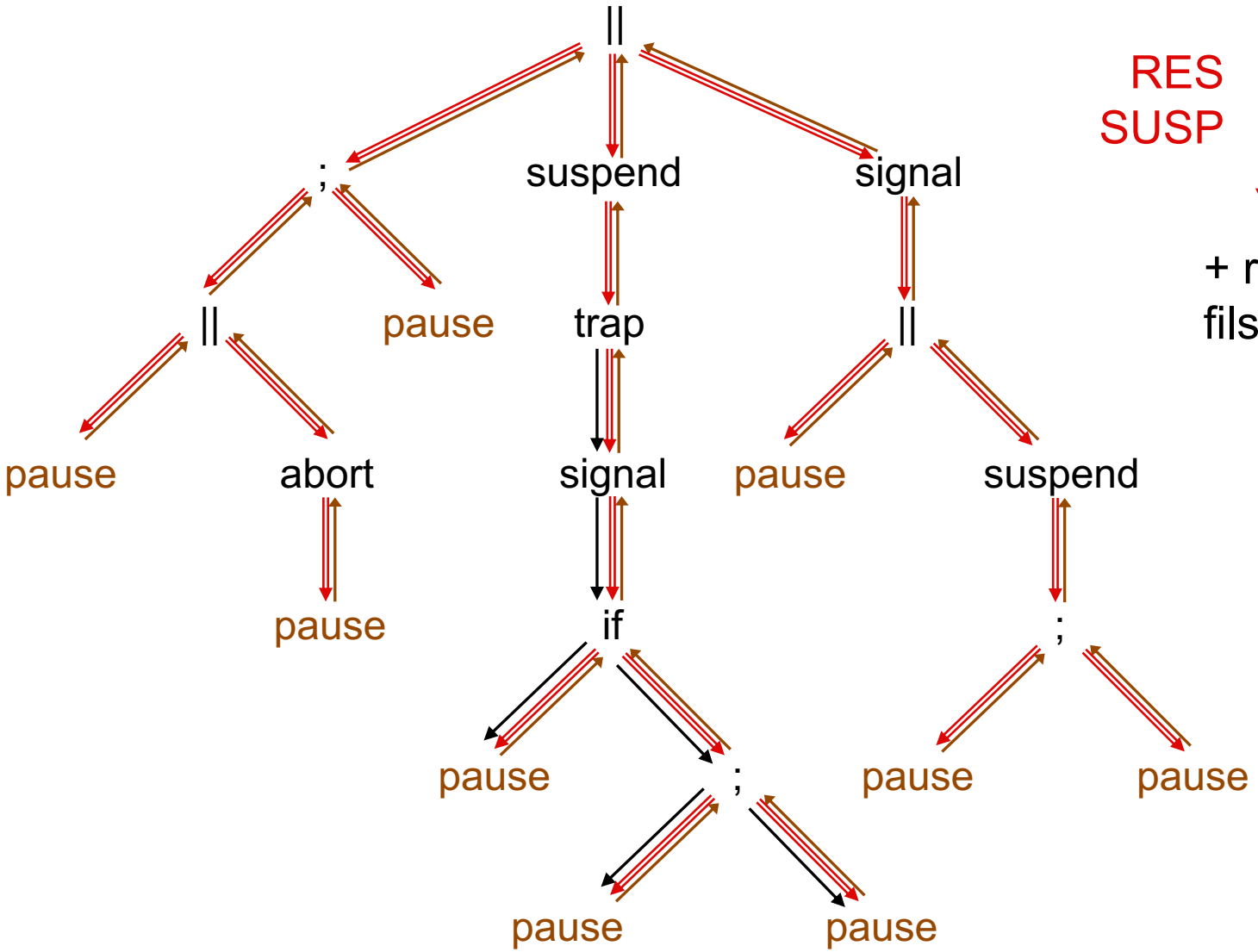
Plan du cours

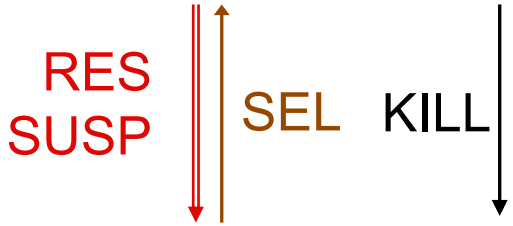
1. Rappels sur les circuits synchrones
2. La sémantique par termes marqués (états)
3. Interface des circuits
4. Circuits de pause, abort, trap et supend
5. Circuits de séquence, et if
6. Circuit de parallèle
7. Circuit de signal
8. L'ossature d'un circuit et les invariants d'états

La syntaxe abstraite d'un terme



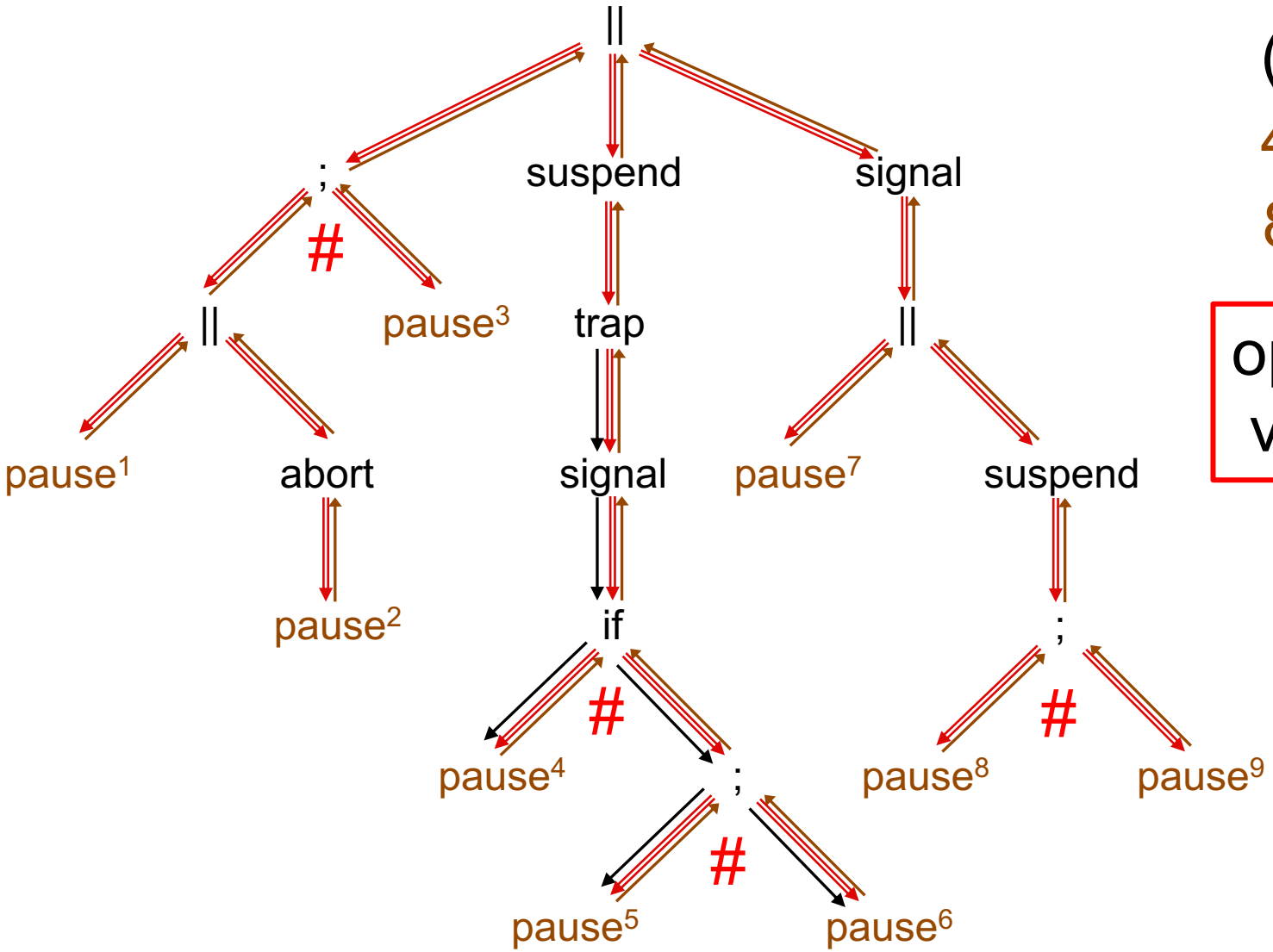
L'ossature d'un circuit





+ réseau de
fils GO, Ki, etc.

Invariant d'incompatbilité des pauses



(1 or 2) # 3

4 # (5 or 6)

8 # 9

optimisation
vérification

Plan du cours

1. Rappels sur les circuits synchrones
2. La sémantique par termes marqués (états)
3. Interface des circuits
4. Circuits de pause, abort, trap et supend
5. Circuits de séquence, et if
6. Circuit de parallèle
7. Circuit de signal
8. Le théorème principal

Liaison sémantique constructives / circuits

Théorème : Pout tout *p* constructif, et toute suite d'entrées pour *p*, le circuit de *p* est électriquement sain et calcule exactement sa sémantique constructive.

Preuve :

1. étendre *Must* et *Can* aux $\hat{p}$
2. montrer que la propagation des 1 calcule *Must*
3. et que la propagation des 0 calcule *Cannot*

Preuve en cours en Coq,
voir le séminaire de Lionel Rieg le 28 mars 2018
Quid de la réciproque (vraie je pense) ?

A suivre au prochain cours

- *Clock-gating* → weak suspend
- Traitement des boucles et le la réincarnation des signaux et instructions