

Μετάφραση σε Υλικό

Διαλέξεις στο μάθημα: Μεταφραστές II
Γεώργιος Μανής

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA



Niklaus Wirth

Swiss Federal Institute of Technology,
Zurich

Hardware Compilation: Translating Programs into Circuits

Computer, vol. 31 issue 6, June 1998

IEEE Computer Society Press

Niklaus Wirth
Swiss Federal Institute of Technology,
Zurich

Hardware Compilation: Translating Programs into Circuits

Niklaus Wirth is a professor at the Swiss Federal Institute of Technology (ETH), Zurich, and is best known for developing Pascal. He received the IEEE Computer Pioneer award in 1988 and the ACM Turing award in 1984. Contact him at wirth@inf.ethz.ch.

Μετάφραση προγραμμάτων σε κυκλώματα

- # Εργαλεία
 - Γλώσσες περιγραφής υλικού
 - **Ειδικές** γλώσσες προγραμματισμού
- # Προτεινόμενη προσέγγιση
 - **Γλώσσες προγραμματισμού**

Γλώσσες σαν εργαλεία κατασκευής υλικού

- # Βασίζεται στη **μείωση** του κόστους υλικού
- # Είναι **δυνατή** ακόμα και η **κατασπατάληση** υλικού
- # Τμήματα του προγράμματος είναι δυνατόν να τρέχουν σε υλικό και άλλα μπορεί να εκτελούνται σε **συμβατικό επεξεργαστή**

Διαφορές υλικού και προγραμματισμού

- ✦ Τα προγράμματα είναι ακολουθίες από εντολές που εκτελούνται **σειριακά**
- ✦ Ένα κύκλωμα αποτελείται από μικρότερα κυκλώματα τα οποία λειτουργούν **παράλληλα** και συνεχώς
- ✦ Τα προγράμματα είναι **δυναμικά** και τα κυκλώματα **στατικά**
- ✦ Στον προγραμματισμό η μετάγλώττιση είναι **το τελευταίο στάδιο**, ενώ στην κατασκευή ενός κυκλώματος όχι, αφού μετά πρέπει να γίνει και η φυσική κατασκευή του

Ομοιότητες

- ⌘ Οι γλώσσες περιγραφής υλικού μοιάζουν με τις γλώσσες προγραμματισμού **αντικαθιστώντας τα σχέδια του υλικού με κείμενο**
 - ⌘ Για κάθε πρόγραμμα ένα κύκλωμα μπορεί να κάνει την **ίδια ακριβώς λειτουργία**
 - ⌘ Τα προγράμματα τελικά **εκτελούνται σε υλικό** και όχι σε μηχανές Turing
 - ⌘ Η **αυτόματη μετάφραση** σε υλικό ενός προγράμματος είναι εφικτή (όχι πάντα όμως και οικονομική)
-

Δήλωση μεταβλητών

VAR x,y,z: Type

Για κάθε μεταβλητή χρησιμοποιείται ένας καταχωρητής

Η τιμή του καταχωρητή είναι και η τιμή της μεταβλητής ενώ η είσοδος enable καθορίζει πότε ο καταχωρητής θα αλλάξει τιμή

Τύποι μεταβλητών

- ✦ Boolean: 1 bit
- ✦ Byte: 8 bits
- ✦ Integer 16 bits
- ✦ Κ.Ο.Κ

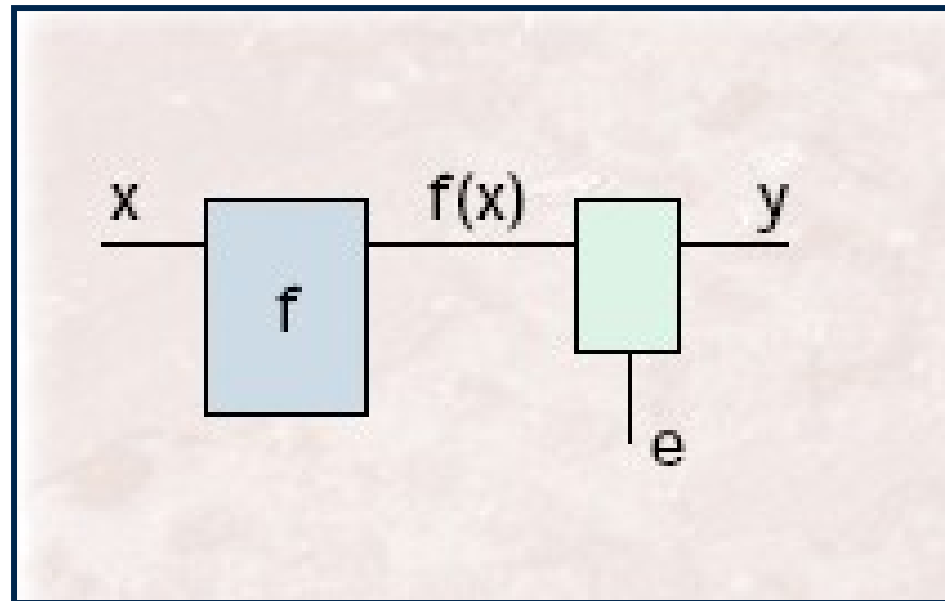
Εκχωρήσεις

$$y=f(x)$$

x ένα σύνολο μεταβλητών

f μία έκφραση του x

y μία μεταβλητή



Εκφράσεις

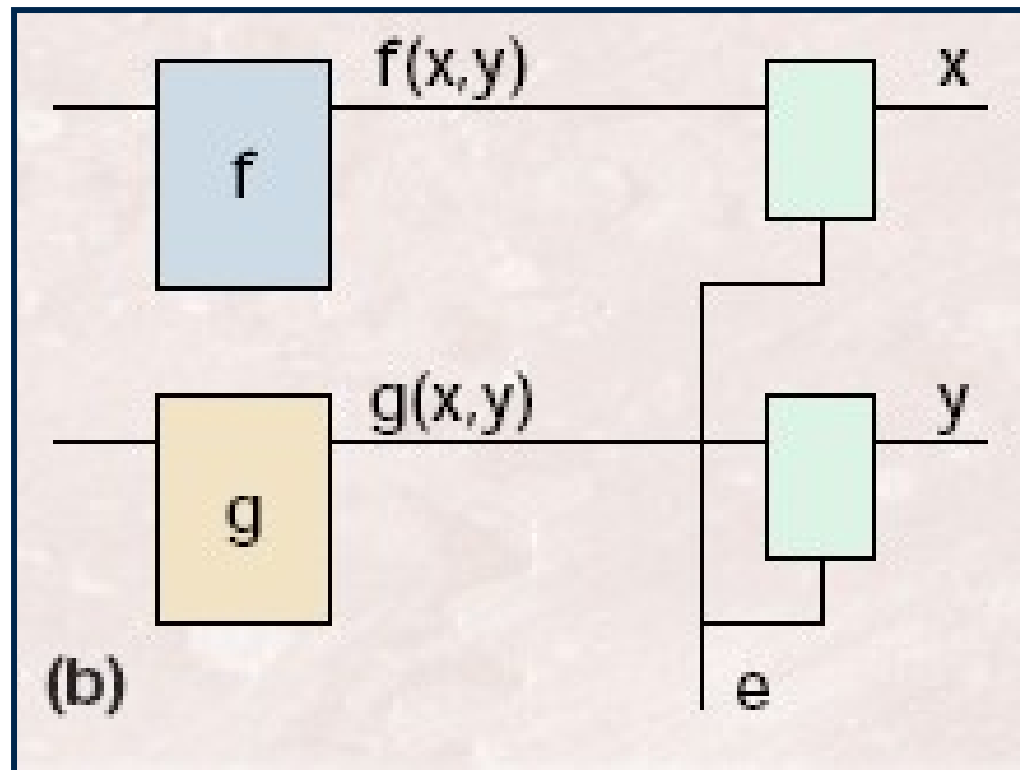
- # Οι μαθηματικές πράξεις υλοποιούνται από **συνδυαστικά ή ακολουθιακά** κυκλώματα
- # **Συνδυασμοί** μαθηματικών πράξεων υλοποιούνται πάλι από ακολουθιακά κυκλώματα, περισσότερο πολύπλοκα
- # **Κάθε έκφραση μπορεί να υπολογιστεί** από ένα συνδυαστικό ή ακολουθιακό κύκλωμα, όσο κι αν αυτό είναι δαπανηρό όταν προκύπτει με αυτόματο τρόπο

Παράλληλη εκτέλεση

So: $x=f(x,y)$

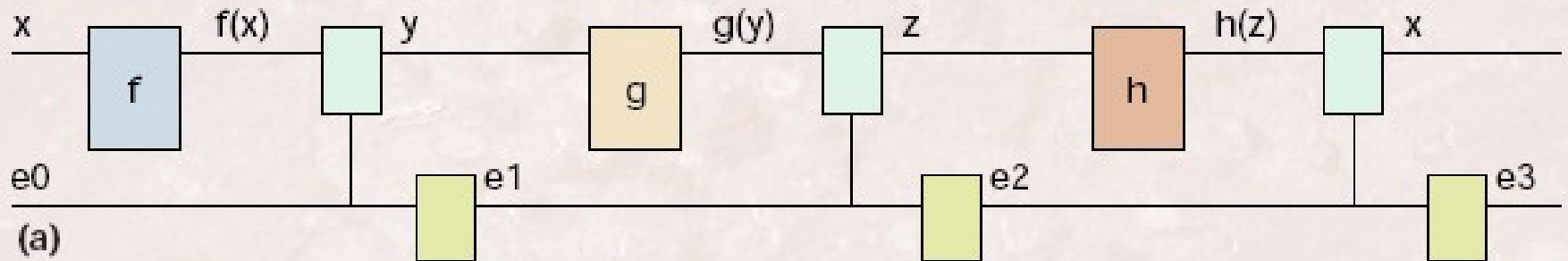
//

S1: $y=g(x,y)$



Σειριακή εκτέλεση

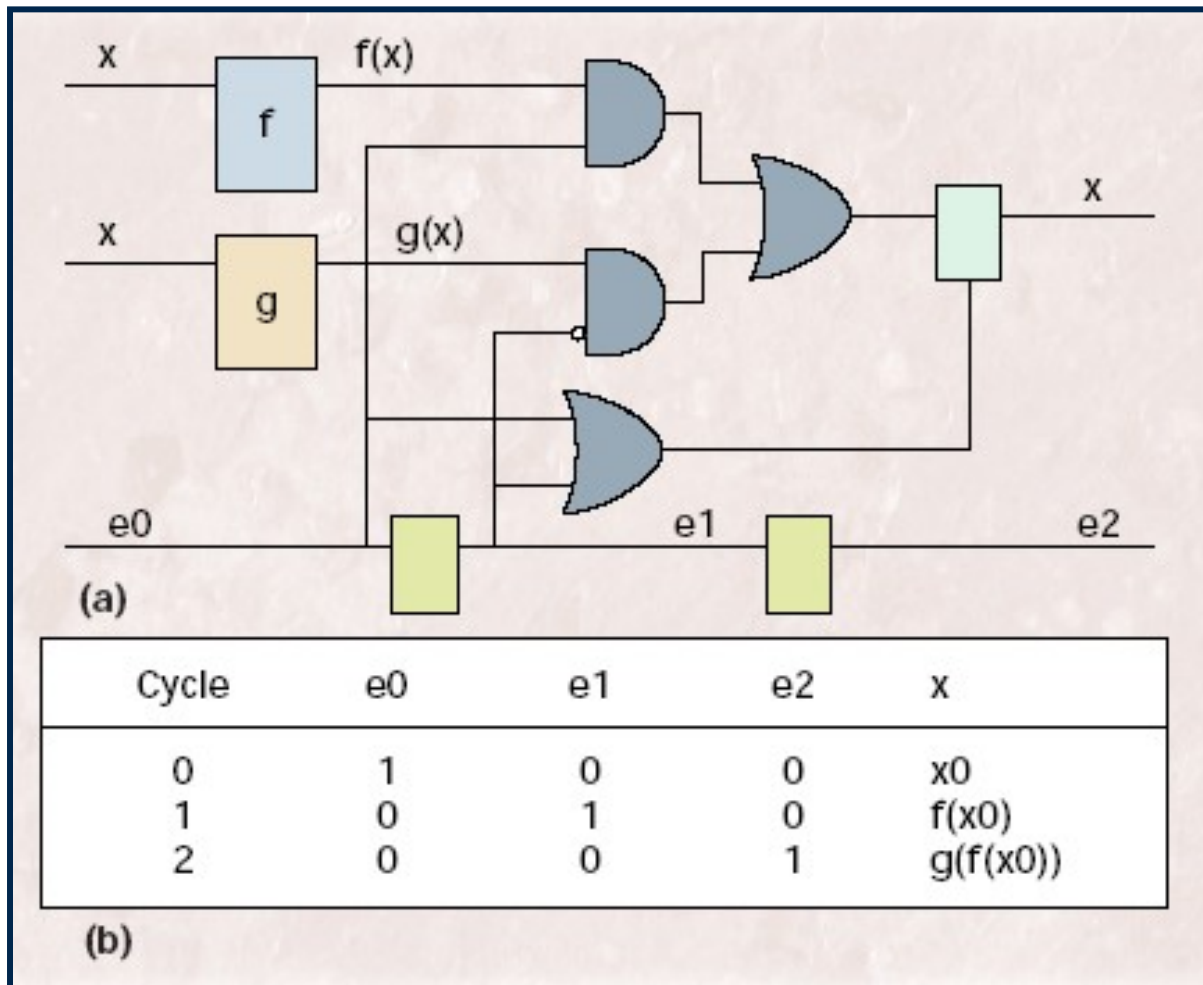
$y := f(x); z := g(y); x := h(z)$



(b)

Cycle	e0	e1	e2	e3	x	y	z
0	1	0	0	0	x0	Unspecified	Unspecified
1	0	1	0	0	x0	f(x0)	Unspecified
2	0	0	1	0	x0	f(x0)	g(f(x0))
3	0	0	0	1	h9g(f(x0)))	f(x0)	g(f(x0))

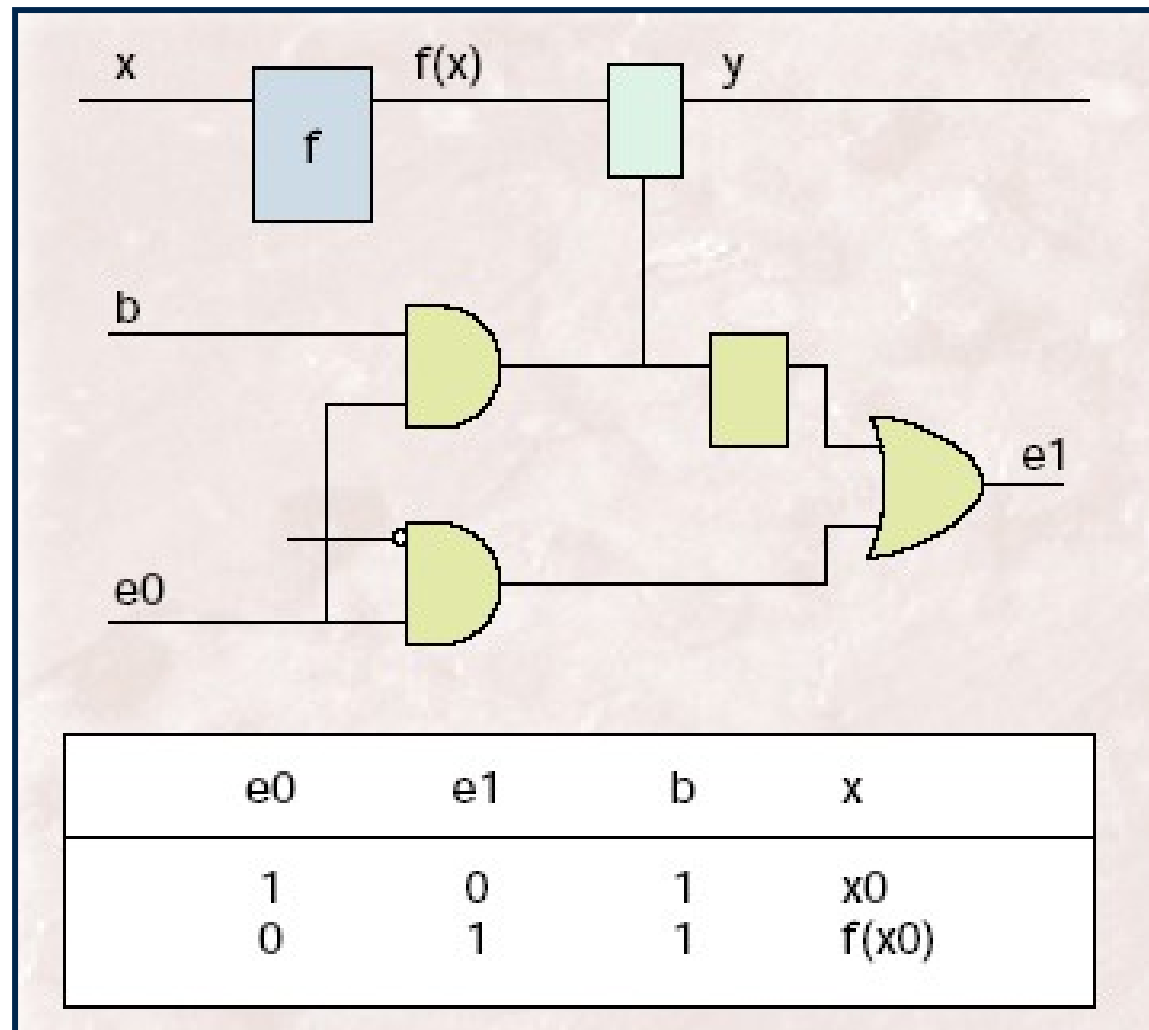
Σειριακή εκτέλεση



Εντολή απόφασης

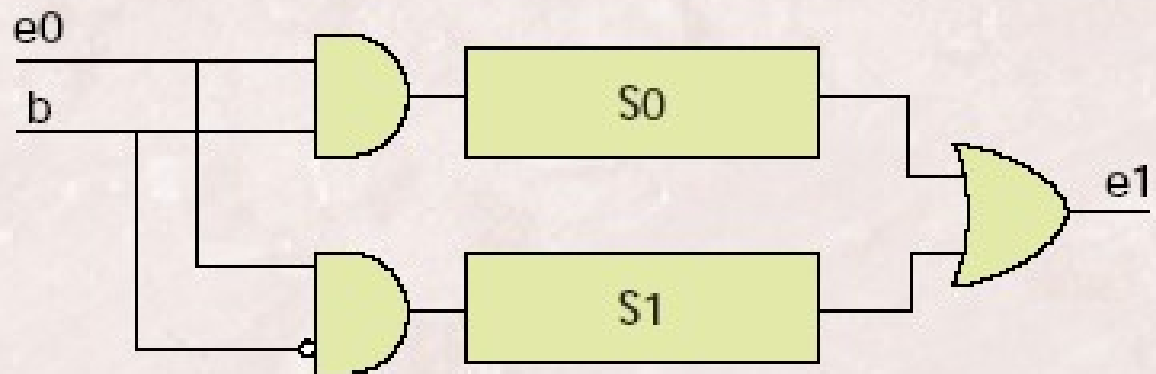
If b then

$$y = f(x)$$



Εντολή απόφασης

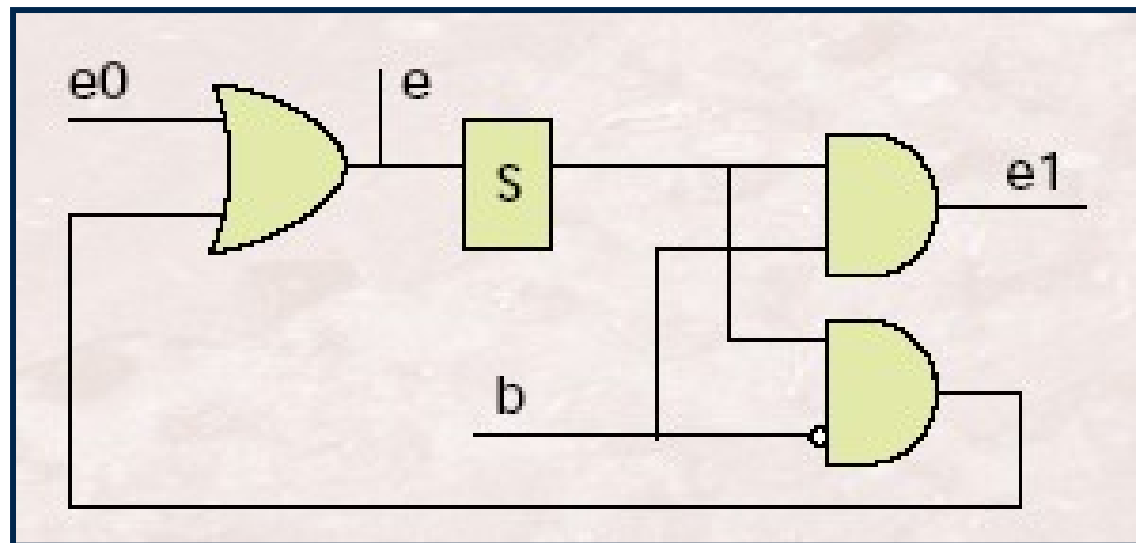
If b
then S0
else S1



<i>e0</i>	<i>e1</i>	<i>b</i>	<i>x</i>
1	1	0	<i>x0</i>

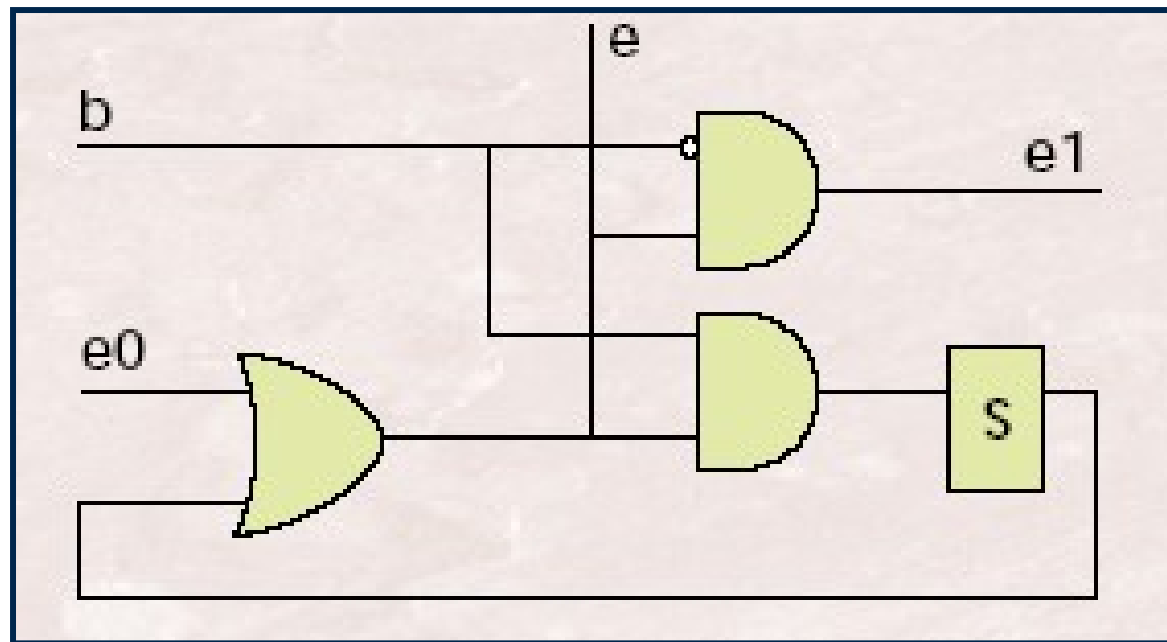
Εντολή επανάληψης

Repeat S
until b



Εντολή επανάληψης

while b
do S



Εντολή Επιλογής

Case k of

0: S0

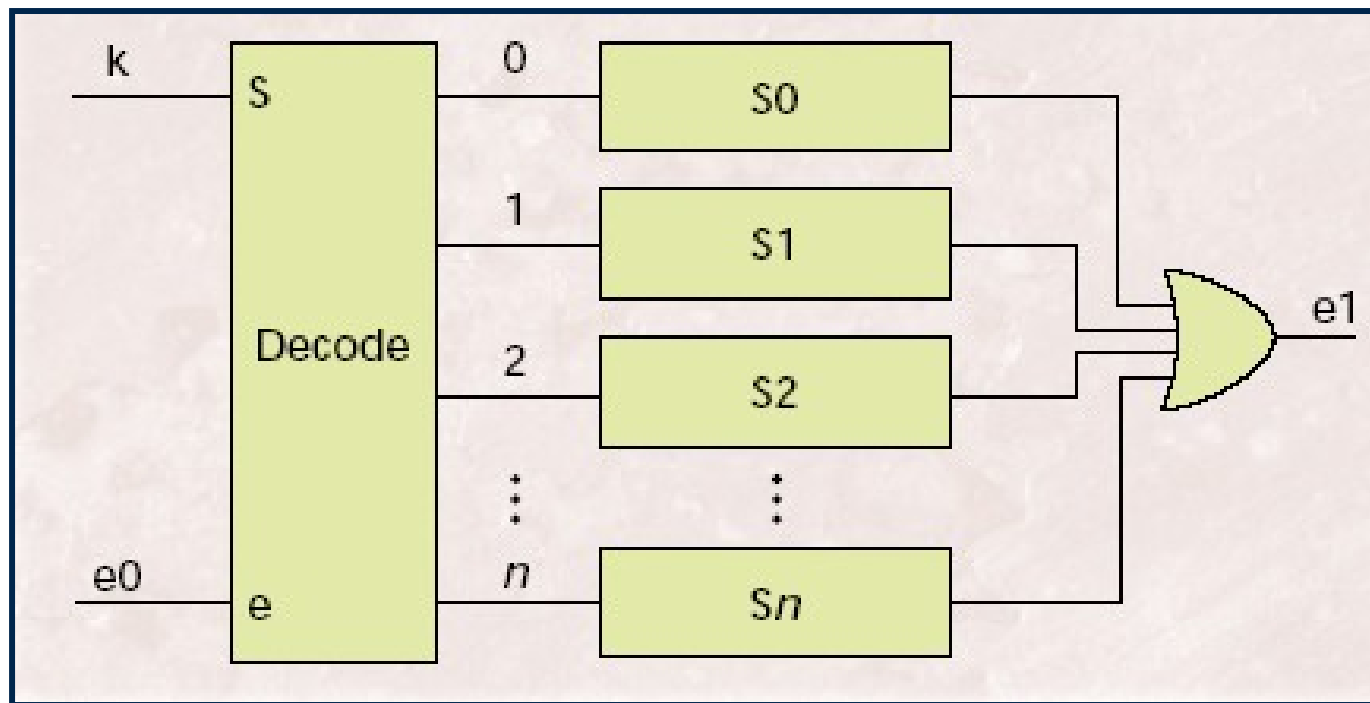
1: S1

2: S2

...

n: Sn

end



Υπορουτίνες

- # **Κλήση** υπορουτίνας
 - Απενεργοποίηση της καλούσας
 - Ενεργοποίηση της κληθείσας
 - Με την ολοκλήρωση της κληθείσας, επιστροφή στην καλούσα
 - # **Υλοποίηση** υπορουτίνας
 - Κυκλώματα ενεργοποίησης και απενεργοποίησης
 - Στοίβα
-

Μία γλώσσα

```
ident = letter {letter | digit}.
integer, digit{digit}.
factor = ident | integer | "TRUE" | "FALSE" | "~" factor | "ODD" factor |
    "(" expression ")" | "{" expression ":" expression "," expression "}".
term = factor {("*" | "/" | "&") factor}.
SimpleExpression = ["+" | "-" | OR) term].
expression = SimpleExpression {("=" | "#" | "<" | ">=" | "<=" | ">")
    Simple Expression}
assignment = ident ":=" expression {"," ident ":=" expression}.
IfStatement = "IF" expression "THEN" StatementSequence ["ELSE"
    StatementSequence] "END".
WhileStatement = "WHILE" expression "DO" StatementSequence
    "END".
Statement = [assignment | IfStatement | WhileStatement].
StatementSequence = Statement {"," Statement}.
type = "BOOLEAN" | "INTEGER".
IdentList = ident {"," ident} ";" type.
declarations = ["CONST" {IdentList ","}] "VAR" {IdentList ","}.
module = "MODULE" ident ";" declaration
    "BEGIN" StatementSequence "END" ident ".".
```

Νούμερα ...

Program module	Data circuits		Sequencer	
	No. of registers	No. of gates	No. of registers	No. of gates
MODULE First; CONST a,b : BOOLEAN; VAR x,y,z : BOOLEAN; BEGIN x := a & b; y := -a OR b; z := a # b END First.	3	7	4	1
MODULE Second; CONST a,b : INTEGER; VAR x,y,z : INTEGER; BEGIN x := a + b, y := a - b, z := a * b END Second.	24	282	2	1
MODULE MinMax; CONST a,b : INTEGER; VAR min,max : INTEGER; BEGIN IF a<b THEN min := a, max := b ELSE min := b, max := a END END MinMax.	16	125	3	5

Άλλα νούμερα

Program module	Data circuits		Sequencer	
	No. of registers	No. of gates	No. of registers	No. of gates
MODULE Log; CONST a,b : INTEGER; VAR x,y : INTEGER; BEGIN x := 0; y := a; WHILE y # 0 DO x := x + 1, y := y / 2 END END Log.	16	105	4	5
MODULE Multiply; CONST a,b : INTEGER VAR x,y,z,n : INTEGER BEGIN n := 8; x := a; y := b; z := 0; WHILE n # 0 DO IF ODD x THEN z := z + y END; x := x / 2, y := y * 2, n := n - 1 END END Multiply.	32	231	4	9

Δυσκολίες

- ✦ Χρησιμοποίηση **μεγάλης ποσότητας υλικού**
 - Βελτιστοποίηση
 - Χρήση υλικού περισσότερες από μία φορές
 - Χρήση υπορουτίνων
- ✦ Η υλοποίηση των **υπορουτίνων** έχει μεγάλη **πολυπλοκότητα**
 - Εκτέλεση τμήματος του προγράμματος σε συμβατικό επεξεργαστή

Ευχαριστώ !!!
