ΗΥ425 - Αρχιτεκτονική Υπολογιστών - Μ4

Χ. Σωτηρίου

16 Οκτωβρίου 2001

Δυναμική Δρομολόγηση Εντολών

- βασικός περιορισμός στις κλιμακώσεις που είδαμε είναι οτι χρησιμοποιούν έκδοση σε-σειρά.
- έτσι άν μια εντολή σταματήσει ούτε και οι επόμενες μπορούν να προχωρήσουν.
- αν έχουμε πολλές μονάδες αυτές μπορεί να μένουν αδρανείς.
- αν μια εντολή j εξαρτάται σε μια i που εκτελείται, οι εντολές μετα την j πρέπει να περιμένουν.

Δυναμική Δρομολόγηση Εντολών

```
DIVD F0, F2, F4
ADDD F10, F0, F8
SUBD F12, F8, F14
```

- η $SUBD$ δεν μπορεί να εκτελεστεί επειδή η εξάρτηση της $ADDD$ στην $DIVD$ προκαλεί σταμαάτημα στην κλιμάκωση.

- έτσι παρόλο που η $SUBD$ είναι αναξέρτητη δεν μπορεί να προχωρήσει.

Δυναμική Δρομολόγηση Εντολών

- στον DLX ο έλεγχος για κινδύνους γίνεται στο ID.
- μπορούμε να χωρίσουμε την διαδικασία έκδοσης σε δύο μερή:
- 1. έλεγχος δομικών κινδύνων.
- 2. αναμονή λόγω κινδύνων δεδομένων.
- ελέγχουμε για δομικούς κινδύνους πρίν εκδόσουμε την εντολή· έκδοση σε-σειρά.
- θέλουμε όμως να ξεκινήσει η εκτέλεση μόλις τα δεδομένα είναι διαθέσιμα - εκτέλεση εκτός -σειράς.

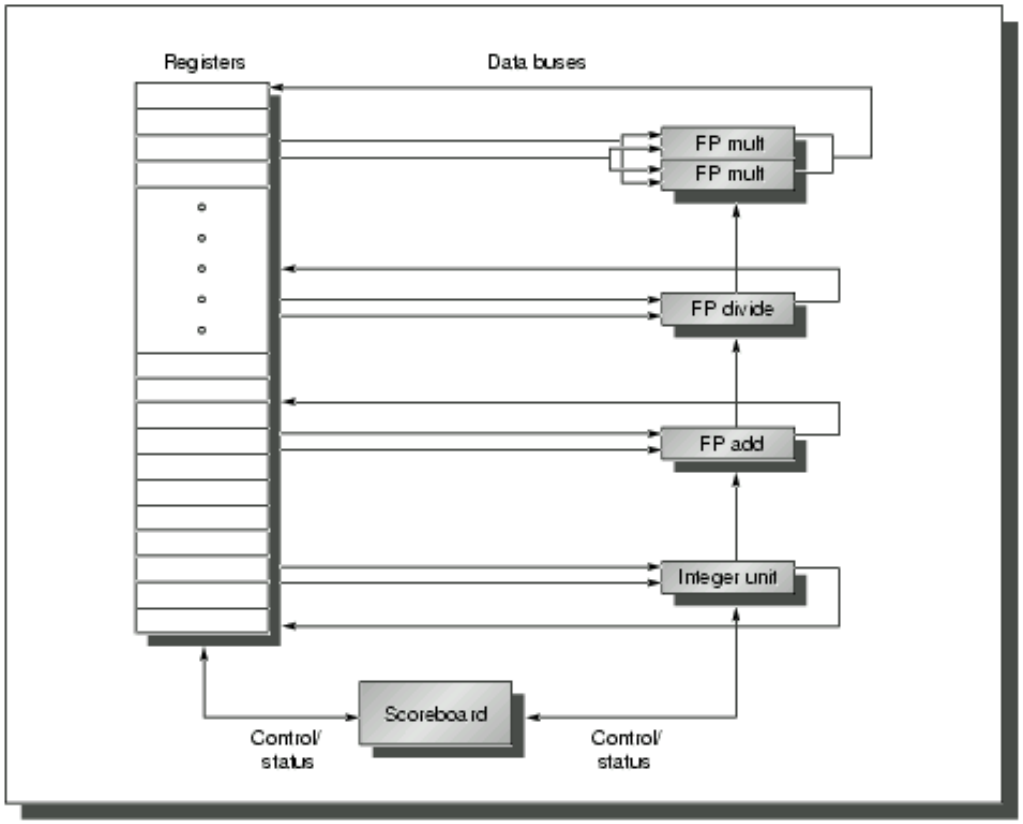
Πίνακας - *Scoreboard*

- τεχνική που επιτρέπει εκτέλεση εκτός -σειράς όταν υπάρχουν αρκε΄τες μονάδες και ελλείψη εξαρτήσεων.
- πρωτοεμφανίστηκε στον *CDC* 6600.
- ο στόχος του πίνακα είναι να διοατηρήσει τον ρυθμό εκτέλεση εντολών σε 1 ανα κύκλο.
- περιέχει όλο το κύκλωμα ελέγχου για έκδοση, εκτέλεση εντολών και χειρισμό κινδύνων.
- η εκτέλεση εκτός -σειράς απαιτεί πολλαπλές μονάδες εκτέλεσης.

Πίνακας - *Scoreboard*

- κάθε εντολή περνά απο τον πίνακα.
- εκέι ελέγχονται οι εξαρτήσεις.
- ο πίνακας αποφασίζει πότε η εντολή μπορεί να διαβάσει τα δρώμενα της και να ξεκινήσει εκτέλεση.
- αν δέν μπορεί να ξεκινήσει αμέσως, ο πίνακας ελέγχει συνεχώς την κατάσταση της μηχανής μέχρι να μπορεί να εκτελεστεί.
- επίσει ελέγχει άν μια εντολή μπορεί να γράψει το αποτέλεσμα.

Η βασική δομή του DLX με Πίνακα ($scoreboard$)



Στάδια εκτέλεσης με πίνακα

Τέσσερα στάδια απαιτεί η εκτέλεση μιας εντολής:

1. Έκδοση - αν η Λ.Μ. για την εντολή είναι ελεύθερη και καμμία ενεργή εντολή δεν έχει τον ίδιο καταχωρητή αποτελέσματος (WAW) τότε η εντολή μπορεί να εκδοθεί.
2. Ανάγνωση Δρώμενων - ο πίνακας παρακολουθέι την διαθεσιμότητα των δρώμενων τα οποία είναι διαθέσιμα αν καμμία ενεργή εντολή δεν πρόκειται να τα γράψει (RAW). Όταν είναι διαθέσιμα η εντολή μπορεί να εκδοθεί.
3. Εκτέλεση - η Λ.Μ. ξεκινά εκτέλεση. Όταν το αποτέλεσμα είναι διαθέσιμο η εντολή ειδοποιεί τον πίνακα.
4. Εγγραφή αποτελεσμάτων - όταν η Λ.Μ. έχει τελειώσει την εκτέλεση μιας εντολής ο πίνακας ελέγχει για κινδύνους WAR και σταματάει η επιτρέπει την εγγραφή του αποτελε΄σματος.

Τα βασικά στοιχεία ενός πίνακα

		Instruction status			
Instruction		**Issue**	**Read operands**	**Execution complete**	**Write result**
LD	F6,34(R2)	√	√	√	√
LD	F2,45(R3)	√	√	√	
MULTD	F0,F2,F4	√			
SUBD	F8,F6,F2	√			
DIVD	F10,F0,F6	√			
ADDD	F6,F8,F2				

	Functional unit status								
Name	**Busy**	**Op**	**Fi**	**Fj**	**Fk**	**Qj**	**Qk**	**Rj**	**Rk**
Integer	Yes	Load	F2	R3				No	
Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

	Register result status								
	F0	**F2**	**F4**	**F6**	**F8**	**F10**	**F12**	**...**	**F30**
FU	Mult1	Integer			Sub	Divide			

Τα βασικά στοιχεία ενός πίνακα

Ο πίνακας αποτελείται απο 3 μερη:

1. Κατάσταση Εντολών, ***Instruction Status*** - δείχνει σε ποιό απο τα τέσσερα στάδια βρίσκεται η εντολή.
2. Κατάσταση Λ.Μ., ***F.U. Status*** - δείχνει την κατάσταση των ***L.M.***. έχει 9 πεδία:
   - ***Busy*** - αν είναι ενεργό η όχι.
   - ***Op*** - λειτουργία που εκτελείται.
   - F_i - καταχωρητής αποτελέσματος.
   - F_j, F_k - πηγαίοι καταχωρητές.
   - Q_j, Q_k - Λ.Μ. που παράγουν F_j, F_k.
   - R_j, R_k - σημαίες που δείχνουν την κατάσταση των F_j, F_k.
3. Κατάσταση αποτελεσμάτων καταχωρητών, ***Register result status*** - δείχνουν ποιά Λ.Μ. θα γράψει τον κάθε καταχωρητή άν μια ενεργή εντολή τον έχει ώς αποτέλεσμα.

Παράδειγμα εκτέλεσης προγράμματος σε πίνακα

	Instruction status			
Instruction	**Issue**	**Read operands**	**Execution complete**	**Write result**
`LD F6,34(R2)`	√	√	√	√
`LD F2,45(R3)`	√	√	√	√
`MULTD F0,F2,F4`	√	√	√	
`SUBD F8,F6,F2`	√	√	√	√
`DIVD F10,F0,F6`	√			
`ADDD F6,F8,F2`	√	√	√	

	Functional unit status								
Name	**Busy**	**Op**	**Fi**	**Fj**	**Fk**	**Qj**	**Qk**	**Rj**	**Rk**
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
Add	Yes	Add	F6	F8	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

	Register result status								
	F0	**F2**	**F4**	**F6**	**F8**	**F10**	**F12**	**...**	**F30**
FU	Mult1			Add		Divide			

Παράδειγμα εκτέλεσης προγράμματος σε πίνακα

	Instruction status			
Instruction	**Issue**	**Read operands**	**Execution complete**	**Write result**
`LD F6,34(R2)`	√	√	√	√
`LD F2,45(R3)`	√	√	√	√
`MULTD F0,F2,F4`	√	√	√	√
`SUBD F8,F6,F2`	√	√	√	√
`DIVD F10,F0,F6`	√	√	√	
`ADDD F6,F8,F2`	√	√	√	√

	Functional unit status								
Name	**Busy**	**Op**	**Fi**	**Fj**	**Fk**	**Qj**	**Qk**	**Rj**	**Rk**
Integer	No								
Mult1	No								
Mult2	No								
Add	No								
Divide	Yes	Div	F10	F0	F6			No	No

	Register result status								
	F0	**F2**	**F4**	**F6**	**F8**	**F10**	**F12**	**...**	**F30**
FU						Divide			

Στάδια εκτέλεσης στον πίνακα

Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result(D)	`Busy(FU)← yes; Op(FU)← op; Fi(FU)← 'D'; Fj(FU)← 'S1'; Fk(FU)← 'S2'; Qj← Result('S1'); Qk← Result('S2'); Rj← not Qj; Rk← not Qk; Result('D')← FU;`
Read operands	Rj and Rk	`Rj← No; Rk← No`
Execution complete	Functional unit done	
Write result	∀*f*((Fj(*f*)≠Fi(FU) or Rj(*f*)=No) & (Fk(*f*) ≠Fi(FU) or Rk(*f*)=No))	`∀f(if Qj(f)=FU then Rj(f)← Yes); ∀f(if Qk(f)=FU then Rk(f)← Yes); Result(Fi(FU))← 0; Busy(FU)← No`

Πίνακας

- η τεχνική του πίνακα χρησιμοποιεί παραλληλισμό σε επίπεδο εντολών (ILP) για να μειώσει το κόστος των σταματημώτων λόγω εξαρτήσεων.

- Η ικανότητα του πίνακα περιορίζεται απο τους παρακάτω παράγοντες:

1. διαθέσιμος παραλληλισμός - ύπαρξη ανεξάρτητων εντολών για εκτέλεση.
2. αριθμός εντολών στον πίνακα - πόσο 'μπροστά' η αρχιτεκτονική ψάχνει για ανεξάρτητες εντολές. Ο αριθμός των εντολών που παρατηρούνται για εκτέλεση λέγεται παράθυρο ($\boldsymbol{window}$).
3. αριθμός και τύποι Λ.Μ. - περιορισμός της δυναμικής δρομολόγησης απο δομικούς κινδύνους.
4. παρουσία αντι-εξαρτήσεων (WAR) και εξαρτήσεων εξόδων (WAW) - οδηγούν σε σταματήματα εντολών.

Αύξηση ILP σε Βρόχους ($Loops$)

Παράδειγμα Βρόχου:

```
for (i=1000; i>0; i=i-1) x[i] = x[i] + s;
```

Σε γλώσσα $assembly$:
(υποθέτουμε `&x[1] = 8`)

```
Loop: LD   F0, 0(R1)          ;F0=array element
      ADDD F4, F0, F2         ;F2=s
      SD   0(R1), F4          ;store result
      SUBI R1, R1, #8         ;decrement pointer
      BNEZ R1, Loop           ;branch R1!=0
```

Αύξηση ILP σε Βρόχους ($Loops$)

Εκτέλεση βρόχου χωρίς στατική δρομολόγηση:

```
                                  Clock cycle issued:
Loop: LD   F0, 0(R1)                        1
      stall                                 2
      ADDD F4, F0, F2                       3
      stall                                 4
      stall                                 5
      SD   0(R1), F4                        6
      SUBI R1, R1, #8                       7
      stall                                 8
      BNEZ R1, Loop                         9
      stall                                10
```

Αύξηση ILP σε Βρόχους ($Loops$)

Στατική δρομολόγηση του βρόχου:

```
                                  Clock cycle issued:
Loop:  LD   F0, 0(R1)                    1
       SUBI R1, R1, #8                   2
       ADDD F4, F0, F2                   3
       stall                             4
       BNEZ R1, Loop                     5
       SD   8(R1), F4                    6
```

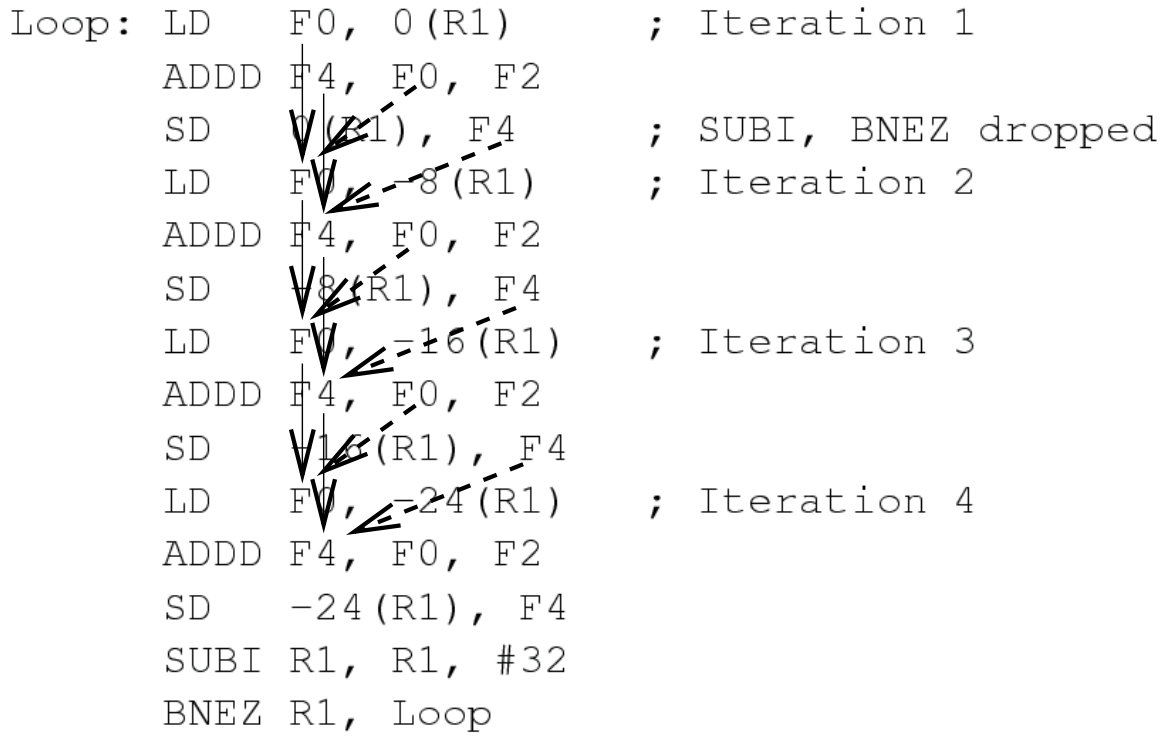
Αύξηση ILP σε Βρόχους ($Loops$)

Ξεδίπλωμα του βρόχου:

```
Loop: LD   F0, 0(R1)     ; Iteration 1
      ADDD F4, F0, F2
      SD   0(R1), F4     ; SUBI, BNEZ dropped
      LD   F0, -8(R1)    ; Iteration 2
      ADDD F4, F0, F2
      SD   -8(R1), F4
      LD   F0, -16(R1)   ; Iteration 3
      ADDD F4, F0, F2
      SD   -16(R1), F4
      LD   F0, -24(R1)   ; Iteration 4
      ADDD F4, F0, F2
      SD   -24(R1), F4
      SUBI R1, R1, #32
      BNEZ R1, Loop
```

Αύξηση ILP σε Βρόχους ($Loops$)

Εξαρτήσεις WAW και WAR στο βρόχο:



Αύξηση *ILP* σε Βρόχους (***Loops***)

- οι εξαρτήσεις *WAW* και *WAR* ονομάζονται ονομαστικές εξαρτήσεις.
- αυτές συμβαίνουν οταν δυο εντολές χρησιμοποιούν τον ίδιο καταχωρητή ή την ίδια διεύθυνση μνήμης αλλα δεν υπάρχει ροή δεδομένων μεταξύ τους.
- οι ονομαστικές εξαρτήσεις υποχρεώνουν τις εντολές να εκτελεστούν με τηνε σειρα.
- μετονομάζοντας τους καταχωρητές μπορούμε να απαλείψουμε τις ονομαστικές εξαρτήσεις.

Αύξηση ILP σε Βρόχους ($Loops$)

Μετονόμασια καταχωρητών:

```
Loop: LD   F0, 0(R1)     ; Iteration 1
      ADDD F4, F0, F2
      SD   0(R1), F4     ; SUBI, BNEZ dropped
      LD   F6, -8(R1)    ; Iteration 2
      ADDD F8, F6, F2
      SD   -8(R1), F8
      LD   F10, -16(R1)  ; Iteration 3
      ADDD F12, F10, F2
      SD   -16(R1), F12
      LD   F14, -24(R1)  ; Iteration 4
      ADDD F16, F14, F2
      SD   -24(R1), F16
      SUBI R1, R1, #32
      BNEZ R1, Loop
```

Αλγόριθμος *Tomasulo*

- τεχνική δυναμικής δρομολόγισης εντολών παρόμοια με τον πίνακα.
- περιλαμβάνει δυναμική μετονομασία των δρώμενων με την χρήση των σταθμών κράτησης (***reservation stations***).
- ο σταθμός κράτησης παίρνει και συγκρατεί ένα δρώμενο, μόλις αυτό είναι διαθέσιμο.
- ετσι δέν απαιτείται απαράιτητα ανάγνωση καταχωρητή.
- επίσεις, τα δρώμενα των εντολών που περιμένουν ορίζουν τους σταθμούς κρατήσεων που θα παράγουν το αποτέλεσμα.

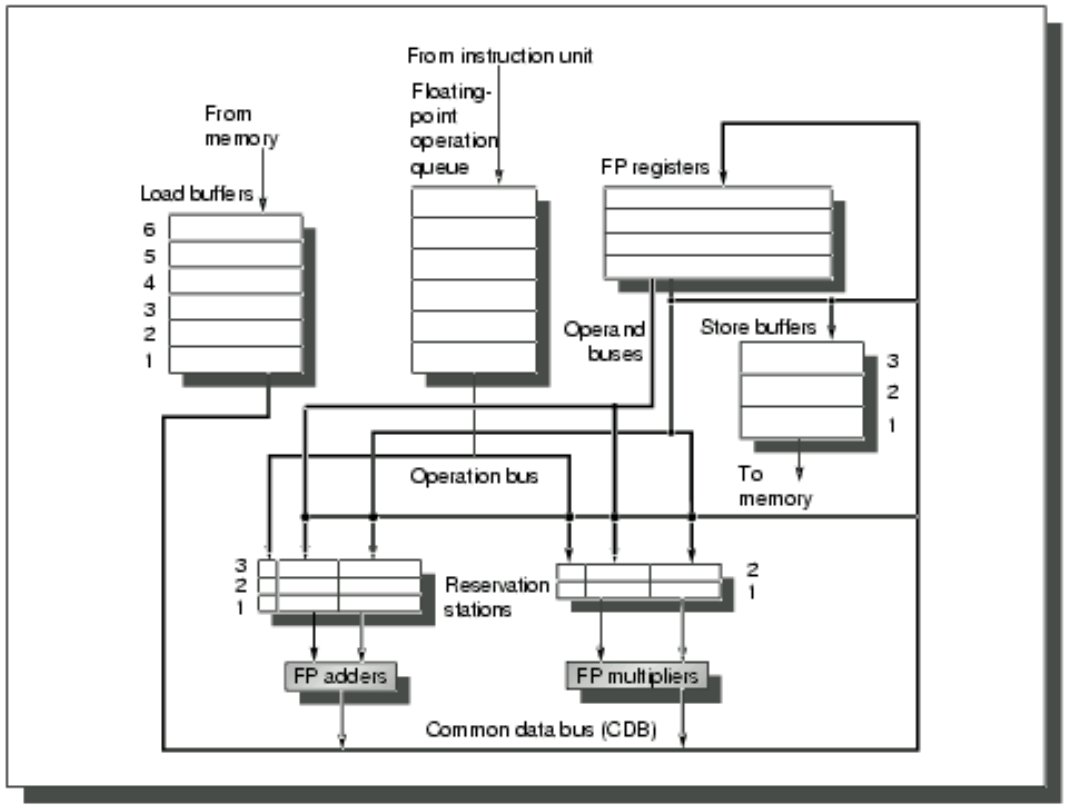
Αλγόριθμος *Tomasulo*

- όταν οι εντολές εκδίδονται, τα πεδία προσδιορισμού των καταχωρητών μετονομάζονται σε ονόματα σταθμών κράτησης.
- ο συνδιασμός λογικής έκδοσης και μετονομασίας είναι η βασική διαφορά απο τον πίνακα.
- η ύπαρξη περισσοτέρων σταθμών κράτησης απο καταχωρητές σημαίνει οτι ο αλγόριθμος *Tomasulo* μπορεί να απαληψεί κινδύνους που ο μεταγλωτιστής δεν μπορεί.

Αλγόριθμος *Tomasulo*

- δυο κύριες διαφορές με τον πίνακα:
- η κατανομή της λογικής κυβώνων και εκτέλεσης.
- στον πίνακα είναι επικεντρωμένη στους τομείς αλγόριθμο *Tomasulo* είναι κατανεμημένη στους τομείς.
- τα αποτελέσματα των εντολών θεν παραδίνε απαραίτητα μέσω κατακχωρητών αλλά μέσω *bus* - του *CDB* (*Common Data Bus*).

Δομή του DLX (FP) που χρησιμοποιεί αλγόριθμο $Tomasulo$.



Στάδια του αλγόριθμου $Tomasulo$

1. Έκδοση:

- η εντολή διαβάζεται απο την ουρά λειτουργιών.
- αν υπάρχει άδειος σταθμός κράτησης εκδίδεται.
- αν τα δρώμενα είναι διαθέσιμα στέλνονται στον σταθμό κράτησης.
- άν η εντολή είναι μνήμης, τότε εκδίδεται αν υπάρχει διαθέσιμος χώρος στους $buffers$.
- αν δεν υπάρχει διαθέσιμος σταθμός κράτησης ή $buffer$ μνήμης τότε έχουμε δομικό κίνδυνο και η εντολή περιμένει.

Στάδια του αλγόριθμου $Tomasulo$

2. Εκτέλεση:

- αν κάποιο απο τα δρώμενα δεν είναι διαθέσιμα το CDB παρακολουθείται μέχρι αυτο να παρουσιαστεί.
- οταν ένα δρώμενο παρουσιαστεί αυτό τοποθετείται στον κατάλληλο σταθμό κράτησης.
- μόλις και τα δύο δρώμενα είναι διαθέσιμα εκτελείται η εντολή.

Στάδια του αλγόριθμου $Tomasulo$

3. Εγγραφή αποτελέσματος:

- όταν το αποτέλεσμα υπολογιστεί γράφεται στο CDB.
- απο το CDB γράφεται στους καταχωρητές και σε κάθε σταθμό κράτησης ή $buffer$ που το χρειάζεται.

Δομή των σταθμών κράτησης και ετικετών καταχωρητών

	Instruction status		
Instruction	**Issue**	**Execute**	**Write result**
`LD F6,34(R2)`	√	√	√
`LD F2,45(R3)`	√	√	
`MULTD F0,F2,F4`	√		
`SUBD F8,F6,F2`	√		
`DIVD F10,F0,F6`	√		
`ADDD F6,F8,F2`	√		

	Reservation stations					
Name	**Busy**	**Op**	**Vj**	**Vk**	**Qj**	**Qk**
Add1	Yes	SUB	Mem[34+Regs[R2]]			Load2
Add2	Yes	ADD			Add1	Load2
Add3	No					
Mult1	Yes	MULT		Regs[F4]	Load2	
Mult2	Yes	DIV		Mem[34+Regs[R2]]	Mult1	

	Register status								
Field	**F0**	**F2**	**F4**	**F6**	**F8**	**F10**	**F12**	**...**	**F30**
Qi	Mult1	Load2		Add2	Add1	Mult2			

Κατάσταση πρίν ολοκληρωθούν οι MUL και DIV

Instruction		Instruction status		
		Issue	Execute	Write result
LD	F6,34(R2)	√	√	√
LD	F2,45(R3)	√	√	√
MULTD	F0,F2,F4	√	√	
SUBD	F8,F6,F2	√	√	√
DIVD	F10,F0,F6	√		
ADDD	F6,F8,F2	√	√	√

	Reservation stations					
Name	Busy	Op	Vj	Vk	Qj	Qk
Add1	No					
Add2	No					
Add3	No					
Mult1	Yes	MULT	Mem[45+Regs[R3]]	Regs[F4]		
Mult2	Yes	DIV		Mem[34+Regs[R2]]	Mult1	

	Register status								
Field	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Mult1					Mult2			

Στάδια στον αλγόριθμο *Tomasulo*

Instruction status	Wait until	Action or bookkeeping
Issue	Station or buffer empty	`if (Register['S1'].Qi ≠0)` `{RS[r].Qj← Register['S1'].Qi}` `else {RS[r].Vj← S1}; RS[r].Qj← 0};` `if (Register[S2].Qi≠0)` `{RS[r].Qk← Register[S2].Qi};` `else {RS[r].Vk← S2; RS[r].Qk← 0}` `RS[r].Busy← yes;` `Register['D'].Qi=r;`
Execute	(RS[r].Qj=0) and (RS[r].Qk=0)	None—operands are in Vj and Vk
Write result	Execution completed at r and CDB available	`∀x(if (Register[x].Qi=r) {Fx← result;` `Register[x].Qi← 0});` `∀x(if (RS[x].Qj=r) {RS[x].Vj← result;` `RS[x].Qj ← 0});` `∀x(if (RS[x].Qk=r) {RS[x].Vk← result;` `RS[x].Qk ← 0});` `∀x(if (Store[x].Qi=r) {Store[x].V← result;` `Store[x].Qi ← 0});` `RS[r].Busy← No`

Εκτέλεση ξεδιπλωμένου βρόχου

		Instruction status			
Instruction		From iteration	Issue	Execute	Write result
LD	F0,0(R1)	1	√	√	
MULTD	F4,F0,F2	1	√		
SD	0(R1),F4	1	√		
LD	F0,0(R1)	2	√	√	
MULTD	F4,F0,F2	2	√		
SD	0(R1),F4	2	√		

	Reservation stations					
Name	Busy	Fm	Vj	Vk	Qj	Qk
Add1	No					
Add2	No					
Add3	No					
Mult1	Yes	MULT		Regs[F2]	Load1	
Mult2	Yes	MULT		Regs[F2]	Load2	

	Register status								
Field	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Load2		Mult2						

	Load buffers		
Field	Load 1	Load 2	Load 3
Address	Regs[R1]	Regs[R1]-8	
Busy	Yes	Yes	No

	Store buffers		
Field	Store 1	Store 2	Store 3
Qi	Mult1	Mult2	
Busy	Yes	Yes	No
Address	Regs[R1]	Regs[R1]-8	