Εκτέλεση Εκτός Σειράς (Out-of-Order)

Μάιος 2023

Ο Χρόνος Ενός Προγράμματος στον CPU…

$$\text{CPU Time} = \boldsymbol{CPU\ clock\ cycles} \times \boldsymbol{clock\ cycle\ time}$$
$$= \frac{\boldsymbol{instructions}}{\boldsymbol{program}} \times \frac{\boldsymbol{cycles}}{\boldsymbol{instruction}} \times \frac{\boldsymbol{seconds}}{\boldsymbol{cycle}}$$

- Βελτίωση (μείωση του χρόνου) αν αυξηθεί η συχνότητα ρολογιού (clock frequency)
- Αν μειωθεί το CPI
  - Pipeline
    - divide CPI by # of stages + bubbles
    - add bypass
  - Superscalar
    - divide CPI by issue width + bubbles from
      - data dependencies
      - control dependencies

Άλλος Τρόπος?

- Τι μας επιβραδύνει εδώ?

```
// for (i=0; i<N; i++) c[i] = a[i]*b[i]+i*2;

loop: ld    r1, (r2+r9*8)
      ld    r3, (r4+r9*8)
      mult  r5, r3, r1
      mult  r3, r9, 2
      add   r5, r5, r3
      st    r5, (r6+r9*8)
      add   r9, r9, 1
      cmp   r9, r10
      bnz   loop
```

3

Άλλος Τρόπος?

- Τι μας επιβραδύνει εδώ?

```
// for (i=0; i<N; i++) c[i] = a[i]*b[i]+i*2;

loop: ld    r1, (r2+r9*8)
      ld    r3, (r4+r9*8)
      mult  r5, r3, r1
      mult  r3, r9, 2
      add   r5, r5, r3
      st    r5, (r6+r9*8)
      add   r9, r9, 1
      cmp   r9, r10
      bnz   loop
```

4

Πως Μπορεί να Βελτιωθεί?

- Έστω τα παρακάτω χαρακτηριστικά του επεξεργαστή:
  - 2-issue superscalar
  - 1 LD/ST unit (2 cycles),
  - 1 mult (2 cycles)
  - 1 add (1 cycle)

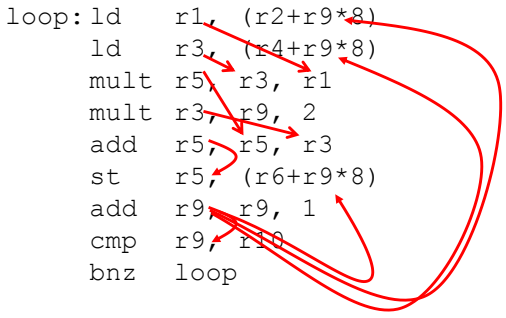
```
ld      r1, (r2+r9*8)
..ld      r3, (r4+r9*8)
  ..mult    r5, r3, r1
    ..mult    r3, r9, 2
      ..add     r5, r5, r3
        .st      r5, (r6+r9*8)
         .add     r9, r9, 1
          .cmp     r9, r10
           .bnz     loop
```

12 cycles

5

Ανάλυση της Εκτέλεσης?

- Τι μας επιβραδύνει?

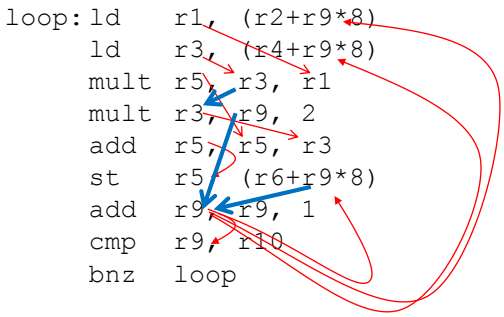


- Data Dependencies
  - RAW: Read after write. (true dependence)

7

Ανάλυση της Εκτέλεσης

- Τι μας επιβραδύνει?

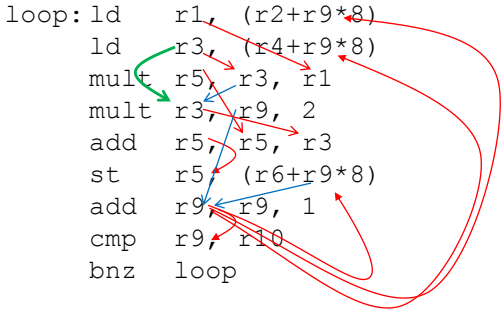


- Data Dependencies
  - RAW: Read after write. (true dependence)
  - WAR: write after read (false dependence)

8

Ανάλυση της Εκτέλεσης

- Τι μας επιβραδύνει?

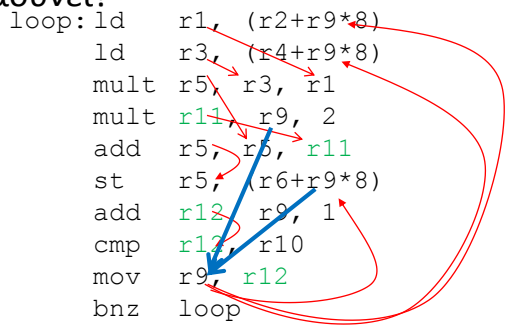


- Data Dependencies
  - RAW: Read after write. (true dependence)
  - WAR: write after read (false dependence)
  - WAW: write after write (false dependence)

9

Απαλοιφή των Εξαρτήσεων WAR (and WAW)

- Τι μας επιβραδύνει?



- Data Dependencies
  - RAW: Read after write. (true dependence)
  - WAR: write after read (false dependence)
  - WAW: write after write (false dependence)

10

Επαναχρονοπρογραμματισμός: Βελτίωση του ILP

```
loop: ld   r1, (r2+r9*8)
      ld   r3, (r4+r9*8)
      mult r5, r3, r1
      mult r11, r9, 2
      add  r5, r5, r11
      st   r5, (r6+r9*8)
      add  r12, r9, 1
      cmp  r12, r10
      mov  r9, r12
      bnz  loop
```

11

Μπορεί να Βελτιωθεί επιπλέον?

```
loop: ld      r1, (r2+r9*8)
      mult    r11, r9, 2
      ..ld      r3, (r4+r9*8)
        ..mult    r5, r3, r1
          ..add     r5, r5, r11
            .st      r5, (r6+r9*8)
             add     r12, r9, 1
             .cmp     r12, r10
              mov     r9, r12
               bnz     loop
```

10 cycles

12

Χρονοπρογραμματισμός Ξανά !

```
loop: ld    r1, (r2+r9*8)
      mult  r11, r9, 2
      ld    r3, (r4+r9*8)
      mult  r5, r3, r1
      add   r5, r5, r11
      st    r5, (r6+r9*8)
      add   r12, r9, 1
      cmp   r12, r10
      mov   r9, r12
      bnz   loop
```

```
loop: ld    r1, (r2+r9*8)
      mult  r11, r9, 2
      ld    r3, (r4+r9*8)
      add   r12, r9, 1
      cmp   r12, r10
      mult  r5, r3, r1
      add   r5, r5, r11
      st    r5, (r6+r9*8)
      mov   r9, r12
      bnz   loop
```

14

Χρονοπρογραμματισμός Ξανά !

```
ld      r1, (r2+r9*8)
mult    r11, r9, 2
..ld      r3, (r4+r9*8)
  add     r12, r9, 1
  .cmp     r12, r10
   mult    r5, r3, r1
   ..add     r5, r5, r11
     .st      r5, (r6+r9*8)
      mov     r9, r12
      .bnz     loop
```

7 cycles

15

Ποιοί Είναι οι Ανασταλτικοί Παράγοντες?

```
ld      r1, (r2+r9*8)
mult    r11, r9, 2
..ld      r3, (r4+r9*8)
  add     r12, r9, 1
  .cmp     r12, r10
   mult    r5, r3, r1
   ..add     r5, r5, r11
     .st      r5, (r6+r9*8)
      mov     r9, r12
      .bnz     loop
```

- In order issue
- Not enough Function units
- Function units too slow
- Not enough register names
- Control dependence
- Static scheduling

16

Ποιοί Είναι οι Ανασταλτικοί Παράγοντες?

```
ld      r1, (r2+r9*8)
mult    r11, r9, 2
..ld      r3, (r4+r9*8)
  add     r12, r9, 1
  .cmp     r12, r10
   mult    r5, r3, r1
   ..add     r5, r5, r11
     .st      r5, (r6+r9*8)
      bnz     loop
       ld      r1, (r2+r12*8)
```

- In order issue
- Not enough Function units
- Function units too slow
- Not enough register names
- Control dependence
- Static scheduling

17

Ποιοί Είναι οι Ανασταλτικοί Παράγοντες?

- In order issue
- Not enough Function units
- Function units too slow
- Not enough register names
- Control dependence
- Static scheduling

```
ld      r1, (r2+r9*8)
..ld      r3, (r4+r9*8)
  ..mult    r5, r3, r1
    ..mult    r3, r9, 2
      ..add     r5, r5, r3
        .st      r5, (r6+r9*8)
         .add     r9, r9, 1
          .cmp     r9, r10
           .bnz     loop
```

Hardware

```
ld      r1, (r2+r9*8)
mult    r11, r9, 2
..ld      r3, (r4+r9*8)
  add     r12, r9, 1
  .cmp     r12, r10
   mult    r5, r3, r1
   ..add     r5, r5, r11
     .st      r5, (r6+r9*8)
      bnz     loop
```

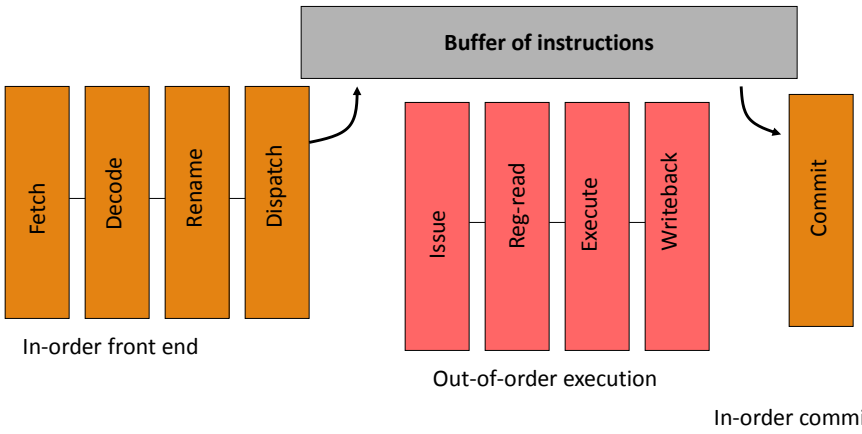
18

Ποιοί Είναι οι Ανασταλτικοί Παράγοντες?

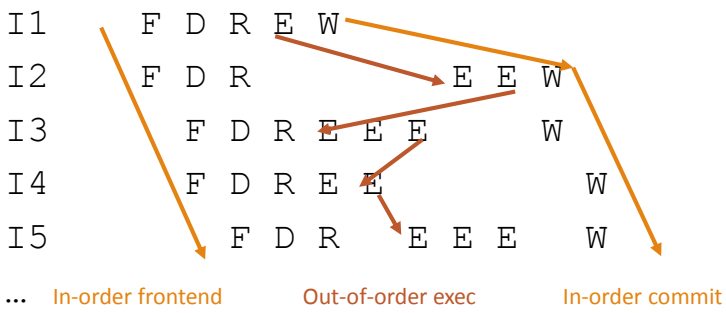
```
ld      r1, (r2+r9*8)
mult    r11, r9, 2
..ld      r3, (r4+r9*8)
  add     r12, r9, 1
  .cmp     r12, r10
   mult    r5, r3, r1
   ..add     r5, r5, r11
     ld      r1, (r2+r12*8)
     .st      r5, (r6+r9*8)
      bnz     loop
```

- In order issue
- Not enough Function units
- Function units too slow
- Not enough register names
- Control dependence
- Static scheduling

Out-of-Order Pipeline



Εκτέλεση Out-of-Order



21

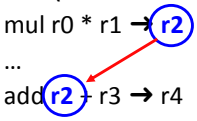
Εκτέλεση Out-of-Order

- Επίσης ονομάζεται Δυναμικός Χρονοπρογραμματισμός (Dynamic scheduling)
  - Εκτελείται από το hardware κατά τη διάρκεια της εκτέλεσης “on-the-fly”
- Ενεργεί πάνω σε ένα “window” από εντολές που είναι έτοιμες για εκτέλεση
  - Σε κάθε κύκλο ρολογιού επιλέγει την επόμενη έτοιμη…
- Για να γίνει η εκτέλεση out-of-order πρέπει να γίνουν τα εξής βήματα:
  Βήμα #1: Επανα-ονοματολογία Καταχωρητών – για την αποφυγή εξαρτήσεων “false”
  Βήμα #2: Δυναμική Χρονοδρομολόγηση – ώστε να γίνουν ακριβείς οι “true” εξαρτήσεις
- Το κλειδί για την κατανόηση της εκτέλεσης out-of-order είναι:
  - **Οι εξαρτήσεις Δεδομένων (Data dependencies)**

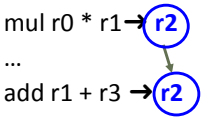
22

Τύποι Εξαρτήσεων

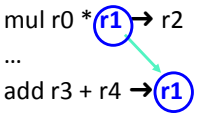
- **RAW** (Read After Write) = “true dependence” (true)



- **WAW** (Write After Write) = “output dependence” (false)



- **WAR** (Write After Read) = “anti-dependence” (false)



*WAW & WAR are “false”, Can be **totally eliminated** by “renaming”*

23

Επανα-ονοματολογία Καταχωρητών

24

Βήμα #1: Επανα-ονοματολογία Καταχωρητών

- Στόχος είναι η εκμηδένιση των register conflicts/hazards
- "Architected" ή "Physical" καταχωρητές – level of indirection
  - Ονόματα: **r1,r2,r3**
  - Θέσεις: **p1,p2,p3,p4,p5,p6,p7**
  - Αρχική απεικόνιση: **r1→p1**, **r2→p2**, **r3→p3**, **p4–p7** είναι "διαθέσιμοι"

MapTable

r1	r2	r3
p1	p2	p3
p4	p2	p3
p4	p2	p5
p4	p2	p6

FreeList

p4,p5,p6,p7
p5,p6,p7
p6,p7
p7

Original insns

```
add r2,r3➔r1
sub r2,r1➔r3
mul r2,r3➔r3
div r1,4➔r1
```

Renamed insns

```
add p2,p3➔p4
sub p2,p4➔p5
mul p2,p5➔p6
div p4,4➔p7
```

  - Επανα-ονοματολογία – στόχος είναι να γραφτεί κάθε καταχωρητής μία φορά !
    - + Αποφεύγονται έτσι οι εξαρτήσεις που είναι **false**
    - + Οι πραγματικές, **true** εξαρτήσεις παραμένουν ανέπαφες!
  - Πότε έπαναχρησιμοποιείται ένας physical register? After overwriting insn done

Επανα-ονοματολογία Καταχωρητών (2)

- Δύο βασικές δομές δεδομένων:
  - maptable[architectural_reg] ➔ physical_reg
  - Free list: allocate (new) & free registers (υλοποιημένες ως ουρά)
- Αλγόριθμος: στο στάδιο "αποκωδικοποίησης" κάθε εντολής:

```
insn.phys_input1 = maptable[insn.arch_input1]
insn.phys_input2 = maptable[insn.arch_input2]
new_reg = new_phys_reg()
maptable[insn.arch_output] = new_reg
insn.phys_output = new_reg
```

35

Μετονομασία - Παράδειγμα

xor r1 ^ r2 ➜ r3
add r3 + r4 ➜ r4
sub r5 - r2 ➜ r3
addi r3 + 1 ➜ r1

r1	p1
r2	p2
r3	p3
r4	p4
r5	p5

Map table

p6
p7
p8
p9
p10

Free-list

36

Μετονομασία – Παράδειγμα (2)

xor r1 ^ r2 ➜ r3 ⟶ xor p1 ^ p2 ➜
add r3 + r4 ➜ r4
sub r5 - r2 ➜ r3
addi r3 + 1 ➜ r1

r1	p1
r2	p2
r3	p3
r4	p4
r5	p5

Map table

p6
p7
p8
p9
p10

Free-list

37

Μετονομασία – Παράδειγμα (3)

xor r1 ^ r2 ➜ r3
add r3 + r4 ➜ r4
sub r5 - r2 ➜ r3
addi r3 + 1 ➜ r1

⟶ xor p1 ^ p2 ➜ p6

r1	p1
r2	p2
r3	p3
r4	p4
r5	p5

Map table

p6
p7
p8
p9
p10

Free-list

38

Μετονομασία – Παράδειγμα (4)

xor r1 ^ r2 ➜ r3
add r3 + r4 ➜ r4
sub r5 - r2 ➜ r3
addi r3 + 1 ➜ r1

⟶ xor p1 ^ p2 ➜ p6

r1	p1
r2	p2
r3	p6
r4	p4
r5	p5

Map table

p7
p8
p9
p10

Free-list

39

Μετονομασία – Παράδειγμα (5)

xor r1 ^ r2 ➜ r3
add r3 + r4 ➜ r4
sub r5 - r2 ➜ r3
addi r3 + 1 ➜ r1

xor p1 ^ p2 ➜ p6
add p6 + p4 ➜

r1	p1
r2	p2
r3	p6
r4	p4
r5	p5

Map table

p7
p8
p9
p10

Free-list

40

Μετονομασία – Παράδειγμα (6)

xor r1 ^ r2 ➜ r3
add r3 + r4 ➜ r4
sub r5 - r2 ➜ r3
addi r3 + 1 ➜ r1

xor p1 ^ p2 ➜ p6
add p6 + p4 ➜ p7

r1	p1
r2	p2
r3	p6
r4	p4
r5	p5

Map table

p7
p8
p9
p10

Free-list

41

Μετονομασία – Παράδειγμα (7)

```
xor r1 ^ r2 ➜ r3
add r3 + r4 ➜ r4
sub r5 - r2 ➜ r3
addi r3 + 1 ➜ r1
```



```
xor  p1 ^ p2 ➜ p6
add p6 + p4 ➜ p7
```

r1	p1
r2	p2
r3	p6
r4	p7
r5	p5

Map table

p8
p9
p10

Free-list

42

Μετονομασία – Παράδειγμα (8)

```
xor r1 ^ r2 ➜ r3
add r3 + r4 ➜ r4
sub r5 - r2 ➜ r3
addi r3 + 1 ➜ r1
```



```
xor  p1 ^ p2 ➜ p6
add p6 + p4 ➜ p7
sub p5 - p2 ➜
```

r1	p1
r2	p2
r3	p6
r4	p7
r5	p5

Map table

p8
p9
p10

Free-list

43

Μετονομασία – Παράδειγμα (9)

```
xor r1 ^ r2 ➜ r3          xor  p1 ^ p2 ➜ p6
add r3 + r4 ➜ r4          add p6 + p4 ➜ p7
sub r5 - r2 ➜ r3   ⟶      sub p5 - p2 ➜ p8
addi r3 + 1 ➜ r1
```

r1	p1
r2	p2
r3	p6
r4	p7
r5	p5

Map table

p8
p9
p10

Free-list

44

Μετονομασία – Παράδειγμα (10)

```
xor r1 ^ r2 ➜ r3          xor  p1 ^ p2 ➜ p6
add r3 + r4 ➜ r4          add p6 + p4 ➜ p7
sub r5 - r2 ➜ r3   ⟶      sub p5 - p2 ➜ p8
addi r3 + 1 ➜ r1
```

r1	p1
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p9
p10

Free-list

45

Μετονομασία – Παράδειγμα (11)

```
xor r1 ^ r2 ➜ r3          xor  p1 ^ p2 ➜ p6
add r3 + r4 ➜ r4          add p6 + p4 ➜ p7
sub r5 - r2 ➜ r3          sub p5 - p2 ➜ p8
addi r3 + 1 ➜ r1   ⟶      addi p8 + 1 ➜
```

r1	p1
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p9
p10

Free-list

46

Μετονομασία – Παράδειγμα (12)

```
xor r1 ^ r2 ➜ r3          xor  p1 ^ p2 ➜ p6
add r3 + r4 ➜ r4          add p6 + p4 ➜ p7
sub r5 - r2 ➜ r3          sub p5 - p2 ➜ p8
addi r3 + 1 ➜ r1   ⟶      addi p8 + 1 ➜ p9
```

r1	p1
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p9
p10

Free-list

47

Μετονομασία – Παράδειγμα (13)

xor r1 ^ r2 ➜ r3
add r3 + r4 ➜ r4
sub r5 - r2 ➜ r3
addi r3 + 1 ➜ r1

xor p1 ^ p2 ➜ p6
add p6 + p4 ➜ p7
sub p5 - p2 ➜ p8
addi p8 + 1 ➜ p9

r1	p9
r2	p2
r3	p8
r4	p7
r5	p5

Map table

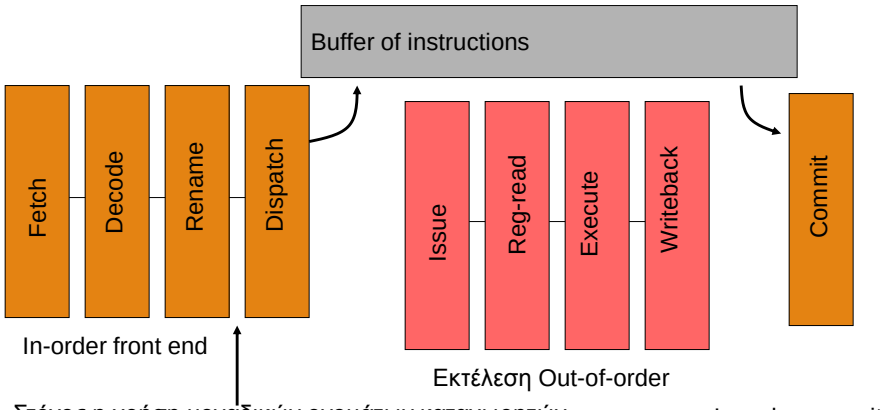
Free-list

48

Μηχανισμοί Δυναμικής Χρονοδρομολόγησης

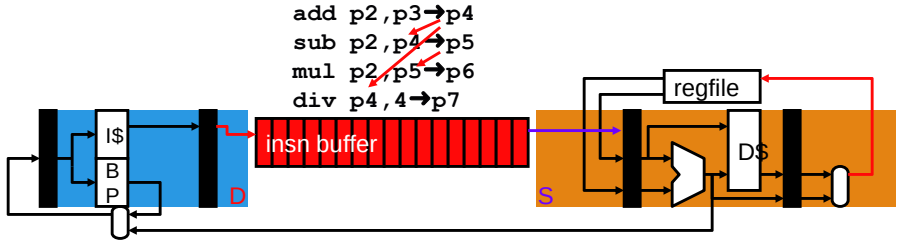
50

Out-of-order Pipeline



51

Βήμα #2: Δυναμική Χρονοδρομολόγηση



Ready Table

Time	P2	P3	P4	P5	P6	P7
	Yes	Yes				
	Yes	Yes	Yes			
	Yes	Yes	Yes	Yes		Yes
	Yes	Yes	Yes	Yes	Yes	Yes

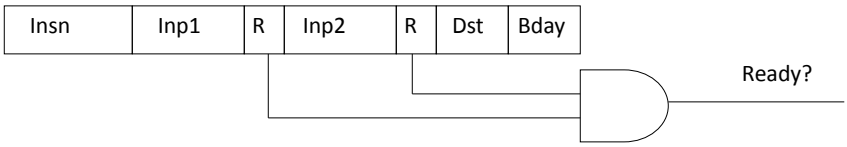
```
add p2,p3→p4
sub p2,p4→p5 and div p4,4→p7
mul p2,p5→p6
```

- Οι εντολές επεξεργάζονται (fetch/decoded/renamed μέσα στον *Instruction Buffer*
  - Επίσης ονομάζεται “instruction window” ή “instruction scheduler”
- Οι εντολές ελέγχουν (conceptually) τα ready bits σε κάθε κύκλο ρολογιού
  - Εκτελείται η πιο παλιά “ready” instruction, και γίνεται set στην έξοδο σαν “ready”

52

Dispatch

- Οι μετονομασμένες εντολές τοποθετούνται σε δομή που ονομάζεται: out-of-order
- Ο Buffer αυτός ονομάζεται Re-order buffer (ROB) και κρατά τις εντολές μέχρι να γίνουν commit
- Issue Queue
  - Βασικό υποσύστημα της λογικής χρονοδρομολόγησης
  - Κρατά τις εντολές που δεν έχουν εκτελεστεί
  - Κρατά ποιες είσοδοι είναι ready
    - Physical register names + ready bit
    - "AND" the bits to tell if ready



53

Βήματα Dispatch

- Allocate Issue Queue (IQ) slot
  - Full? Stall
- Read **ready bits** of inputs
  - 1-bit per physical reg
- Clear **ready bit** of output in table
  - Instruction has not produced value yet
- Write data into Issue Queue (IQ) slot

54

Dispatch Example

xor p1 ^ p2 → p6
add p6 + p4 → p7
sub p5 - p2 → p8
addi p8 + 1 → p9

Issue Queue

Insn	Inp1	R	Inp2	R	Dst	Bday

Ready bits

p1	y
p2	y
p3	y
p4	y
p5	y
p6	y
p7	y
p8	y
p9	y

55

Dispatch Example

xor p1 ^ p2 → p6
add p6 + p4 → p7
sub p5 - p2 → p8
addi p8 + 1 → p9

Issue Queue

Insn	Inp1	R	Inp2	R	Dst	Bday
xor	p1	y	p2	y	p6	0

Ready bits

p1	y
p2	y
p3	y
p4	y
p5	y
p6	**n**
p7	y
p8	y
p9	y

56

Dispatch Example

xor p1 ^ p2 → p6
add p6 + p4 → p7
sub p5 - p2 → p8
addi p8 + 1 → p9

Issue Queue

Insn	Inp1	R	Inp2	R	Dst	Bday
xor	p1	y	p2	y	p6	0
add	p6	n	p4	y	p7	1

Ready bits

p1	y
p2	y
p3	y
p4	y
p5	y
p6	n
p7	**n**
p8	y
p9	y

57

Dispatch Example

xor p1 ^ p2 → p6
add p6 + p4 → p7
sub p5 - p2 → p8
addi p8 + 1 → p9

Issue Queue

Insn	Inp1	R	Inp2	R	Dst	Bday
xor	p1	y	p2	y	p6	0
add	p6	n	p4	y	p7	1
sub	p5	y	p2	y	p8	2

Ready bits

p1	y
p2	y
p3	y
p4	y
p5	y
p6	n
p7	n
p8	**n**
p9	y

58

Dispatch Example

xor p1 ^ p2 → p6
add p6 + p4 → p7
sub p5 - p2 → p8
addi p8 + 1 → p9

Issue Queue

Insn	Inp1	R	Inp2	R	Dst	Bday
xor	p1	y	p2	y	p6	0
add	p6	n	p4	y	p7	1
sub	p5	y	p2	y	p8	2
addi	p8	n	---	y	p9	3

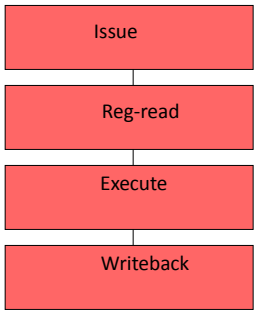
Ready bits

p1	y
p2	y
p3	y
p4	y
p5	y
p6	n
p7	n
p8	n
p9	**n**

59

Out-of-order pipeline

- Execution (out-of-order) stages
- **Select** ready instructions
  - Send for execution
- **Wakeup** dependents



60

Αλγόριθμος Issue

- Δομές δεδομένων:
  - Ready table[phys_reg] ➔ yes/no (part of "issue queue")
- Αλγόριθμος στο στάδιο "issue" (προτού διαβαστούν οι καταχωρητές):

```
foreach instruction:
    if table[insn.phys_input1] == ready &&
       table[insn.phys_input2] == ready then
           insn is "ready"
select the oldest "ready" instruction
    table[insn.phys_output] = ready
```

- Εντολές με πολλαπλούς κύκλους ρολογιού? (όπως τα loads)
  - For an insn with latency of N, set "ready" bit N-1 cycles in future

61

Issue = Επιλογή + Εκκίνηση !

- **Επιλογή** των παλαιότερων εντολών που είναι έτοιμες "ready"
  - ➢ Single-issue: "xor" is the oldest ready instruction below
  - ➢ Dual-issue: "xor" and "sub" are the two oldest ready instructions below
  - Παρατήρηση: ενδέχεται να υπάρχουν structural hazards (i.e., load/store/floating point)

Insn	Inp1	R	Inp2	R	Dst	Bday	
xor	p1	y	p2	y	p6	0	Ready!
add	p6	n	p4	y	p7	1	
sub	p5	y	p2	y	p8	2	Ready!
addi	p8	n	---	y	p9	3	

63

Issue = Επιλογή + Εκκίνηση !

- Εκκίνηση εξαρτώμενων εντολών
  - Εύρεση του προορισμού (Dst) στις στήλες των εισόδων & set “ready” bit
    - Υλοποιείται με ειδική μνήμη: Content Addressable Memory (CAM)
  - Επίσης γίνεατι update ready-bit table για μελλοντικές εντολές

Insn	Inp1	R	Inp2	R	Dst	Bday
xor	p1	y	p2	y	**p6**	0
add	**p6**	**y**	p4	y	p7	1
sub	p5	y	p2	y	**p8**	2
addi	**p8**	**y**	---	y	p9	3

Ready bits

p1	y
p2	y
p3	y
p4	y
p5	y
p6	**y**
p7	n
p8	**y**
p9	n

- Σε λειτουργίες που απαιτούν πολλαπλούς κύκλους (loads, floating point)
  - Η εκκίνηση (Wakeup) μετατίθεται μετικούς κύκλους
  - Περιλαμβάνει ελέγχους για την αποφυγή structural hazards

64

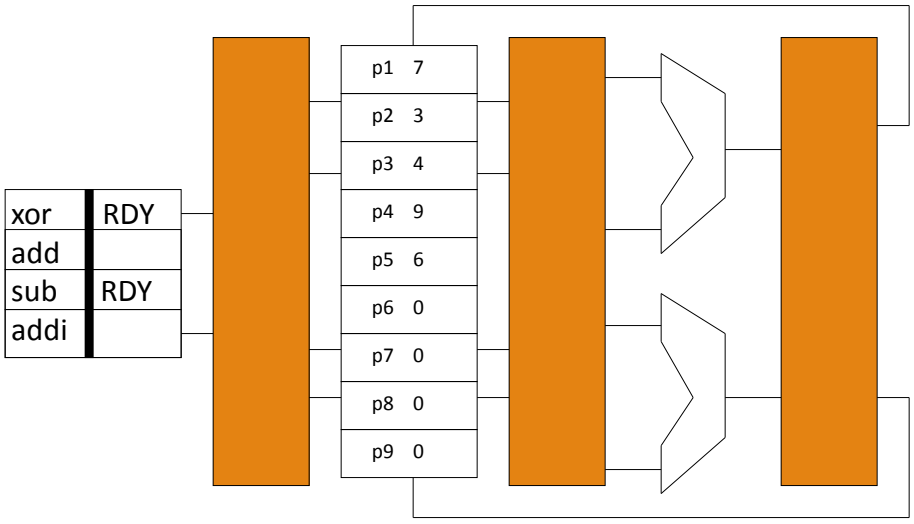
Issue

- **Επιλογή/εκκίνηση** σε ένα κύκλο ρολογιού
- Οι εντολές με εξαρτήσεις εκτελούνται σε κύκλους ρολογιού back-to-back
  - Next cycle: add/addi are ready:
- Οι εντολές που έχουν επιλεγεί βγαίνουν εκτός από την ουρά issue

Insn	Inp1	R	Inp2	R	Dst	Bday
add	p6	**y**	p4	y	p7	1
addi	p8	**y**	---	y	p9	3

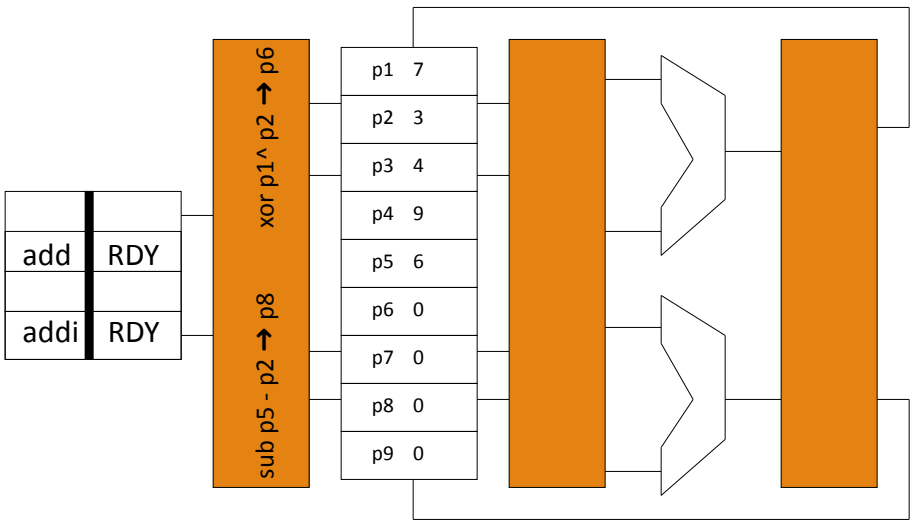
65

OOO execution (2-wide)



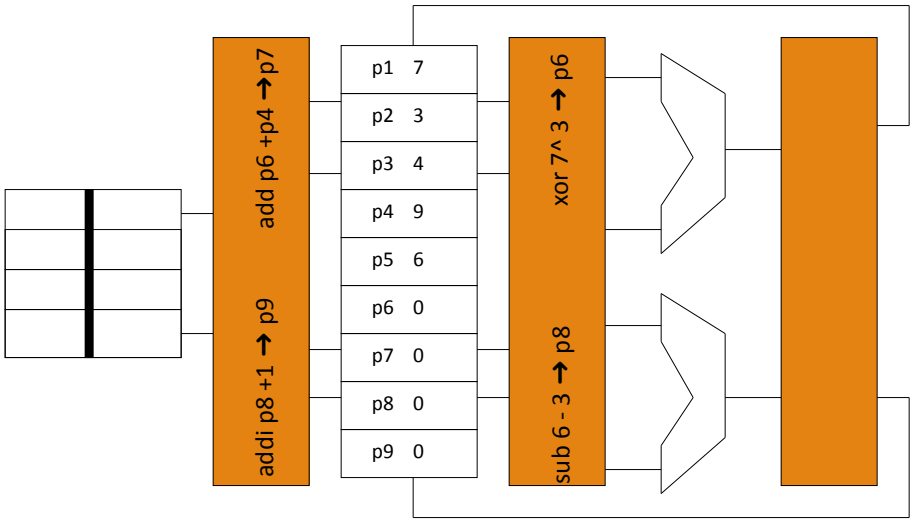
66

OOO execution (2-wide)



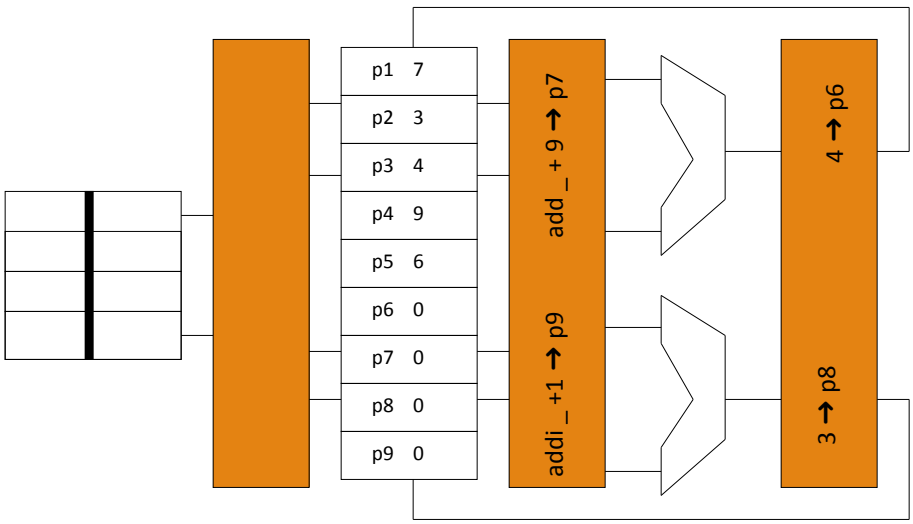
67

OOO execution (2-wide)



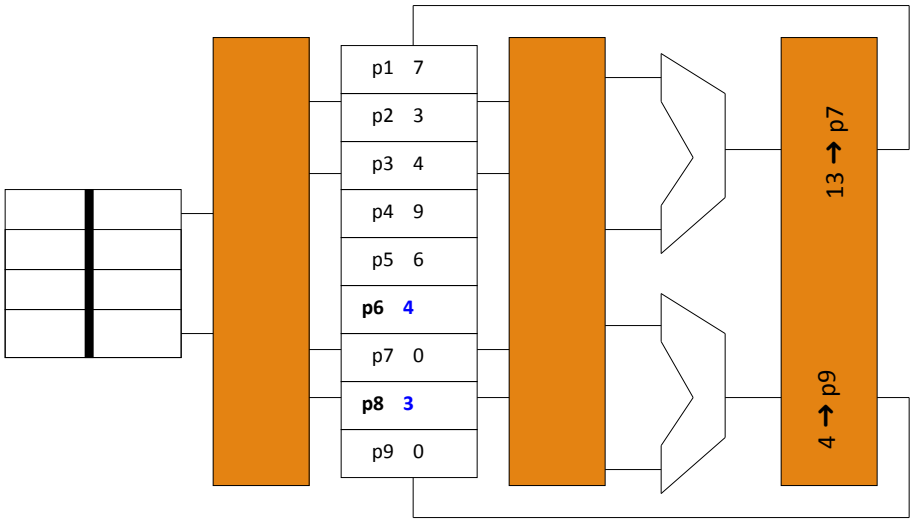
68

OOO execution (2-wide)



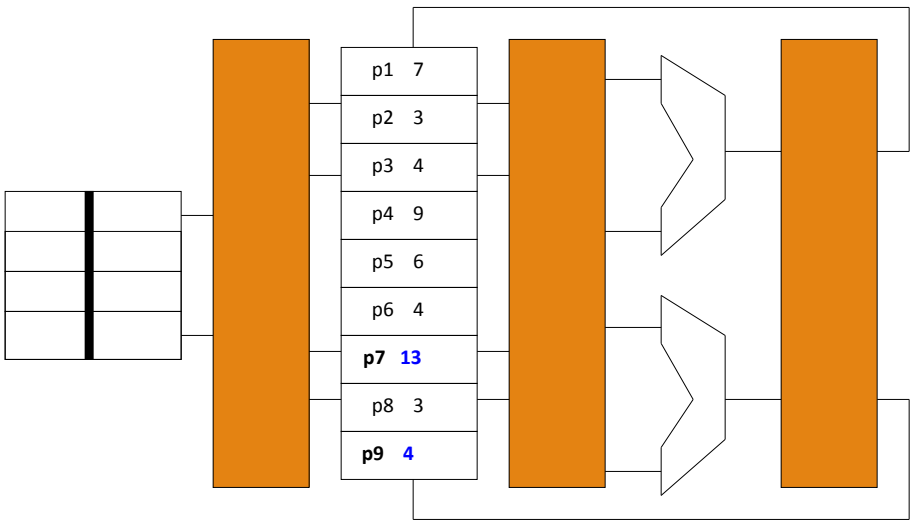
69

OOO execution (2-wide)



70

OOO execution (2-wide)



71

Πότε Γίνεται το Διάβασμα των Καταχωρητών?

- Τωρινή λύση: μετά την επιλογή, ακριβώς πριν την εκτέλεση
  - **ΟΧΙ κατά τη φάση της εκτέλεσης που θα γινόταν in-order (στο στάδιο του out-of-order)**
  - Διάβασμα του καταχωρητή **physical** (του μετονομασμένου καταχωρητή)
  - Ή χρήση της τιμής μέσω bypassing (με βάση το όνομα του physical καταχωρητή)
  - Αυτό γίνεται σε: Pentium 4, MIPS R10k, Alpha 21264, IBM Power4, Intel's "Sandy Bridge" (2011)
  - Το Physical register file μπορεί να είναι μεγάλο ! ➔ απαιτεί διάβασμα multi-cycle
- Παλαιότερη λύση:
  - Το διάβασμα γινόταν στο στάδιο "issue", και αποθηκεύονταν οι τιμές στην ουρά: Issue Queue
    - Στο στάδιο commit, αποθηκεύονταν πίσω στο "architectural register file"
  - Pentium Pro, Core 2, Core i7
  - απλούστερο, αλλά καταναλώνεται περισσότερη ενέργεια (περισσότερες μετακινήσεις data)

72

Μετονομασία – μέρος #2

73

Re-order Buffer (ROB)

- ROB κρατά πληροφορία για recovery/commit
  - **Για όλες τις εντολές πληροφορίες** & και με σειρά !
  - Τα ονόματα καταχωρητών (architectural, physical), και τύπο εντολής
  - Η πληροφορίες αυτές αποθηκεύονται μέχρι **τέλους** (“commit”)

- Λειτουργίες:
  - Dispatch: εισαγωγή στο tail (if full, stall)
  - Commit: εξαγωγή από το head (if not yet done, stall)

- Στόχος: οδηγίες για να γίνει σωστά το in-order commit
  - Είναι απαραίτητος για δύο λειτουργίες:
  - Επανάκαμψη από λάθος πρόβλεψη
  - Ελευθέρωση των καταχωρητών physical registers

74

Παράδειγμα Μετονομασίας

xor r1 ^ r2 → r3
add r3 + r4 → r4
sub r5 - r2 → r3
addi r3 + 1 → r1

r1	p1
r2	p2
r3	p3
r4	p4
r5	p5

Map table

p6
p7
p8
p9
p10

Free-list

78

Παράδειγμα Μετονομασίας

xor r1 ^ r2 ➔ r3
add r3 + r4 ➔ r4
sub r5 - r2 ➔ r3
addi r3 + 1 ➔ r1

⟶ xor p1 ^ p2 ➔ [p3]

r1	p1
r2	p2
r3	**p3**
r4	p4
r5	p5

Map table

p6
p7
p8
p9
p10

Free-list

79

Παράδειγμα Μετονομασίας

xor r1 ^ r2 ➔ r3
add r3 + r4 ➔ r4
sub r5 - r2 ➔ r3
addi r3 + 1 ➔ r1

⟶ xor p1 ^ p2 ➔ p6 [p3]

r1	p1
r2	p2
r3	**p6**
r4	p4
r5	p5

Map table

p7
p8
p9
p10

Free-list

80

Παράδειγμα Μετονομασίας

```
xor r1 ^ r2 → r3                xor  p1 ^ p2 → p6        [ p3 ]
add r3 + r4 → r4      ⟶         add p6 + p4 →            [ p4 ]
sub r5 - r2 → r3
addi r3 + 1 → r1
```

r1	p1
r2	p2
r3	p6
r4	**p4**
r5	p5

Map table

p7
p8
p9
p10

Free-list

81

Παράδειγμα Μετονομασίας

```
xor r1 ^ r2 → r3                xor  p1 ^ p2 → p6        [ p3 ]
add r3 + r4 → r4      ⟶         add p6 + p4 → p7         [ p4 ]
sub r5 - r2 → r3
addi r3 + 1 → r1
```

r1	p1
r2	p2
r3	p6
r4	**p7**
r5	p5

Map table

p8
p9
p10

Free-list

82

Παράδειγμα Μετονομασίας

xor r1 ^ r2 → r3
add r3 + r4 → r4
sub r5 - r2 → r3
addi r3 + 1 → r1

xor p1 ^ p2 → p6
add p6 + p4 → p7
sub p5 - p2 →

[p3]
[p4]
[p6]

r1	p1
r2	p2
r3	**p6**
r4	p7
r5	p5

Map table

p8
p9
p10

Free-list

83

Παράδειγμα Μετονομασίας

xor r1 ^ r2 → r3
add r3 + r4 → r4
sub r5 - r2 → r3
addi r3 + 1 → r1

xor p1 ^ p2 → p6
add p6 + p4 → p7
sub p5 - p2 → p8

[p3]
[p4]
[p6]

r1	p1
r2	p2
r3	**p8**
r4	p7
r5	p5

Map table

p9
p10

Free-list

84

Παράδειγμα Μετονομασίας

xor r1 ^ r2 → r3	xor p1 ^ p2 → p6	[p3]
add r3 + r4 → r4	add p6 + p4 → p7	[p4]
sub r5 - r2 → r3	sub p5 - p2 → p8	[p6]
addi r3 + 1 → r1	addi p8 + 1 →	[p1]

r1	**p1**
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p9

p10

Free-list

85

Παράδειγμα Μετονομασίας

xor r1 ^ r2 → r3	xor p1 ^ p2 → p6	[p3]
add r3 + r4 → r4	add p6 + p4 → p7	[p4]
sub r5 - r2 → r3	sub p5 - p2 → p8	[p6]
addi r3 + 1 → r1	addi p8 + 1 → p9	[p1]

r1	**p9**
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p10

Free-list

86

Παράδειγμα Μετονομασίας

Επανάκαμψη από μία λάθος πρόβλεψη

bnz r1 loop	**bnz p1, loop**	[ ]
xor r1 ^ r2 → r3	xor p1 ^ p2 → p6	[p3]
add r3 + r4 → r4	add p6 + p4 → p7	[p4]
sub r5 - r2 → r3	sub p5 - p2 → p8	[p6]
addi r3 + 1 → r1	addi p8 + 1 → p9	[p1]

r1	p9
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p10

Free-list

87

Παράδειγμα Μετονομασίας

bnz r1 loop	bnz p1, loop	[ ]
xor r1 ^ r2 → r3	xor p1 ^ p2 → p6	[p3]
add r3 + r4 → r4	add p6 + p4 → p7	[p4]
sub r5 - r2 → r3	sub p5 - p2 → p8	[p6]
addi r3 + 1 → r1	addi p8 + 1 → p9	[p1]

r1	**p1**
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p9
p10

Free-list

88

Παράδειγμα Μετονομασίας

bnz r1 loop	bnz p1, loop	[ ]
xor r1 ^ r2 → r3	xor p1 ^ p2 → p6	[p3]
add r3 + r4 → r4	add p6 + p4 → p7	[p4]
sub r5 - r2 → r3	sub p5 - p2 → p8	[p6]

r1	p1
r2	p2
r3	**p6**
r4	p7
r5	p5

Map table

p8
p9
p10

Free-list

89

Παράδειγμα Μετονομασίας

bnz r1 loop	bnz p1, loop	[ ]
xor r1 ^ r2 → r3	xor p1 ^ p2 → p6	[p3]
add r3 + r4 → r4	add p6 + p4 → p7	[p4]

r1	p1
r2	p2
r3	p6
r4	**p4**
r5	p5

Map table

p7
p8
p9
p10

Free-list

90

Παράδειγμα Μετονομασίας

bnz r1 loop
xor r1 ^ r2 → r3

bnz p1, loop
xor p1 ^ p2 → p6

[ ]
[p3]

r1	p1
r2	p2
r3	**p3**
r4	p4
r5	p5

Map table

p6
p7
p8
p9
p10

Free-list

91

Παράδειγμα Μετονομασίας

bnz r1 loop

bnz p1, loop

[ ]

r1	p1
r2	p2
r3	p3
r4	p4
r5	p5

Map table

p6
p7
p8
p9
p10

Free-list

92

Commit

xor r1 ^ r2 → r3	xor p1 ^ p2 → p6	[p3]
add r3 + r4 → r4	add p6 + p4 → p7	[p4]
sub r5 - r2 → r3	sub p5 - p2 → p8	[p6]
addi r3 + 1 → r1	addi p8 + 1 → p9	[p1]

- Commit: η εντολή θα γίνει στην αρχική της κατάσταση:**architected state**
  - In-order, μόνο όταν όλες οι εντολές τελειώσουν
  - ελευθέρωση των καταχωρητών overwritten

93

Ελευθέρωση Καταχωρητών over-written

xor r1 ^ r2 → r3	xor p1 ^ p2 → p6	[p3]
add r3 + r4 → r4	add p6 + p4 → p7	[p4]
sub r5 - r2 → r3	sub p5 - p2 → p8	[p6]
addi r3 + 1 → r1	addi p8 + 1 → p9	[p1]

- Ο P3 ήταν r3 πριν το xor
- P6 είναι r3 μετά το xor
  - οτιδήποτε παλαιότερο από το xor πρέπει να διαβάσει p3
  - οτιδήποτε νεώτερο από το xor πρέπει να διαβάζει p6 (μέχρι μια άλλη εντολή να γράψει το r3)
- Όταν γίνει commit το xor, δεν υπάρχει παλαιότερη εντολή

94

Commit

xor r1 ^ r2 → r3	xor p1 ^ p2 → p6	[p3]
add r3 + r4 → r4	add p6 + p4 → p7	[p4]
sub r5 - r2 → r3	sub p5 - p2 → p8	[p6]
addi r3 + 1 → r1	addi p8 + 1 → p9	[p1]

r1	p9
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p10

Free-list

95

Commit

xor r1 ^ r2 → r3	xor p1 ^ p2 → p6	[p3]
add r3 + r4 → r4	add p6 + p4 → p7	[p4]
sub r5 - r2 → r3	sub p5 - p2 → p8	[p6]
addi r3 + 1 → r1	addi p8 + 1 → p9	[p1]

r1	p9
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p10

p3

Free-list

96

Commit

add r3 + r4 ➔ r4	add p6 + p4 ➔ p7	[p4]
sub r5 - r2 ➔ r3	sub p5 - p2 ➔ p8	[p6]
addi r3 + 1 ➔ r1	addi p8 + 1 ➔ p9	[p1]

r1	p9
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p10
p3
p4

Free-list

97

Commit

sub r5 - r2 ➔ r3	sub p5 - p2 ➔ p8	[p6]
addi r3 + 1 ➔ r1	addi p8 + 1 ➔ p9	[p1]

r1	p9
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p10
p3
p4
p6

Free-list

98

Commit

addi r3 + 1 ➔ r1 addi p8 + 1 ➔ p9 [p1]

r1	p9
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p10
p3
p4
p6
p1

Free-list

99

Commit

r1	p9
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p10
p3
p4
p6
p1

Free-list

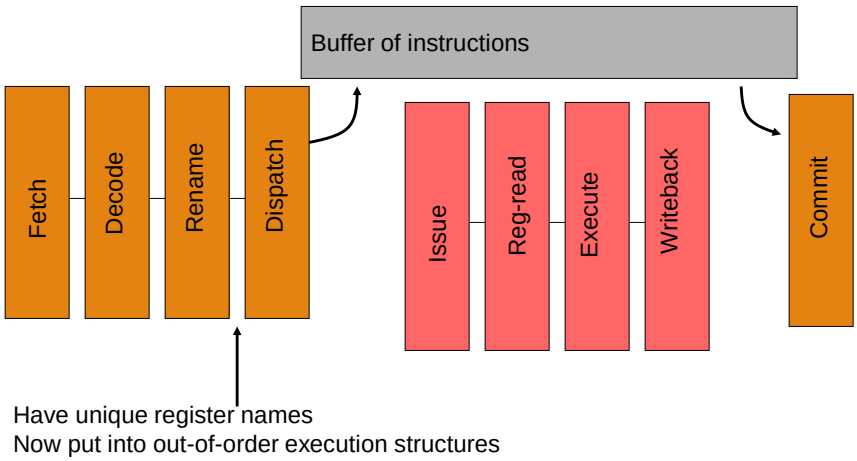
100

Loads και stores

- Δεν γίνονται οι εντολές αποθήκευσης issue μέχρις ότου η εντολή κάνει commit (γιατί όχι?)
- Η δομή δεδομένων Load-store queue (LSQ) κρατάει τα stores που εκτελούνται
- Οι εντολές load που ακολουθούν πρέπει να ελέγξουν την LSQ, και να πάρουν την τιμή που είναι youngest earlier
  - περίπλοκο associative lookup

101

Out-of-order Pipeline



102

OOO: Πλεονεκτήματα & Προκλήσεις

OOO Λειτουργίες (συνοπτικά)

- Διάβασμα πολλών εντολών μέσα σε ένα παράθυρο εντολών
  - Χρήση πρόβληψης διακλαδώσεων για να γίνουν speculate past (multiple) branches
  - Σε περίπτωση λάθους πρόβλεψης σε διακλάδωση τότε κάνε Flush pipeline
- Μετονομασία εντολών για αποφυγή false εξαρτήσεων
- Εκτέλεση εντολών
  - Οι εξαρτήσεις καταχωρητών είναι γνωστές
  - Η διαχείριση των εξαρτήσεων μνήμης είναι δυσκολότερη (vs compiler)
- Εκτέλεση των “Commit” in order
  - Οτιδήποτε περίεργο εμφανιστεί πριν το commit, απλά γίνεται flush το pipeline

Out of Order: Πλεονεκτήματα

- OOO είναι πολύ καλό στο να κρύβει **short** latencies
  - Big IPC gain vs in-order superscalar
- Πρόβλεψη εκτέλεσης (Speculation)
  - Βοηθά σε λειτουργίες μεγάλης καθυστέρησης (loads, divides) σε διακλαδώσεις
  - Καλύτερη δρομολόγηση code (schedule) από ότι μπορεί ένας compiler (often even in principle!)
- Δυναμική δρομολόγηση
  - Τροποποίηση δρομολόγησης code schedule όταν συναντά λειτουργίες με μεταβλητή καθυστέρηση
- Ποιό είναι πιο σημαντικό?
  - Η εκτέλεση με πρόβλεψη δίνει >80% of the benefit

105

Out of Order: Πλεονεκτήματα

- Scheduling done in hardware!
  - Compilers must re-schedule code for different microarch in in-order
  - OOO “just works”
- ➔ OOO runs “bad code” well
  - Code compiled for old machine
  - Pointer-chasing code hard for compilers to optimize

106

Προκλήσεις για Πυρήνες Out-of-Order

- Πολυπλοκότητα σχεδίασης
  - More complicated than in-order? Certainly!
  - But, we have managed to overcome the design complexity
- Συχνότητα ρολογιού
  - Can we build a "high ILP" machine at high clock frequency?
  - Yep, with some additional pipe stages, clever design
- Όρια για (αποδοτική) επέκταση του παραθύρου εντολών και παραλληλισμού (ILP)
  - Large physical register file
  - Fast register renaming/wakeup/select/load queue/store queue
    - Active areas of micro-architectural research
  - Branch & memory depend. prediction (limits effective window size)
    - 95% branch mis-prediction: 1 in 20 branches, or 1 in 100 insn.
  - Plus all the issues of building "wide" in-order superscalar
- Αποδοτικότητα σε κατανάλωση ισχύος
  - Today, even mobile phone chips are out-of-order cores

107

A Brief Survey of MicroArch Complexity

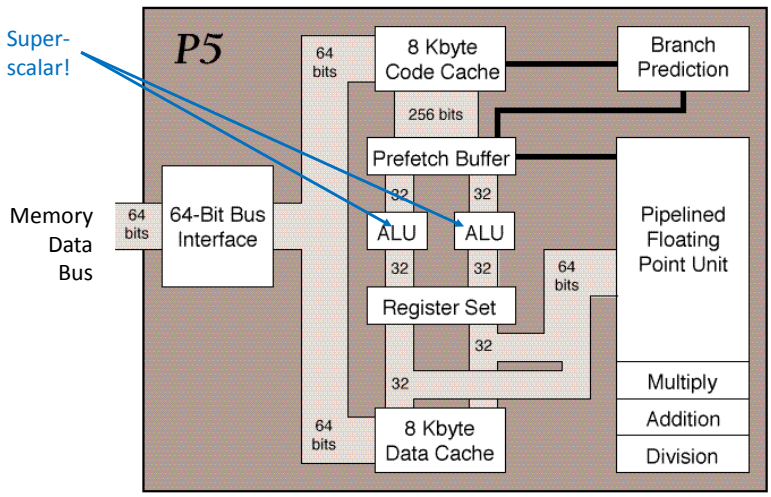
108

i486 Pipeline

- Fetch
  - Load 16-bytes of instruction into prefetch buffer
- Decode1
  - Determine instruction length, instruction type
- Decode2
  - Compute memory address
  - Generate immediate operands
- Execute
  - Register Read
  - ALU operation
  - Memory read/write
- Write-Back
  - Update register file

109

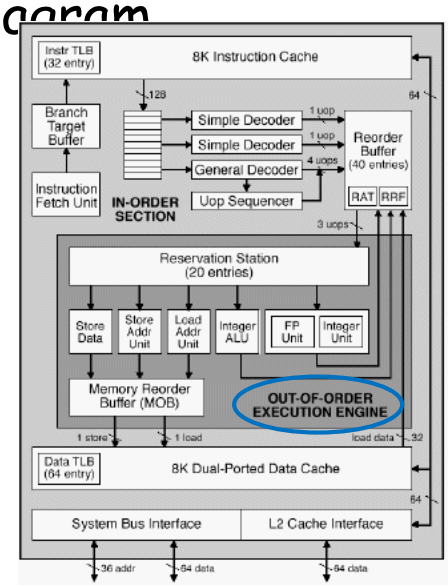
Pentium Block Diagram



(Microcprocessor Report 10/28/92)

116

PentiumPro Block Diagram

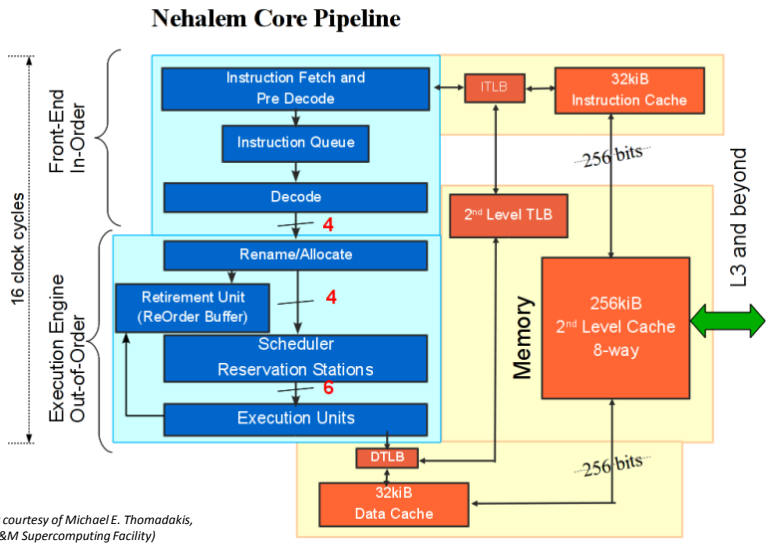


Microprocessor Report
2/16/95

– ## –

121

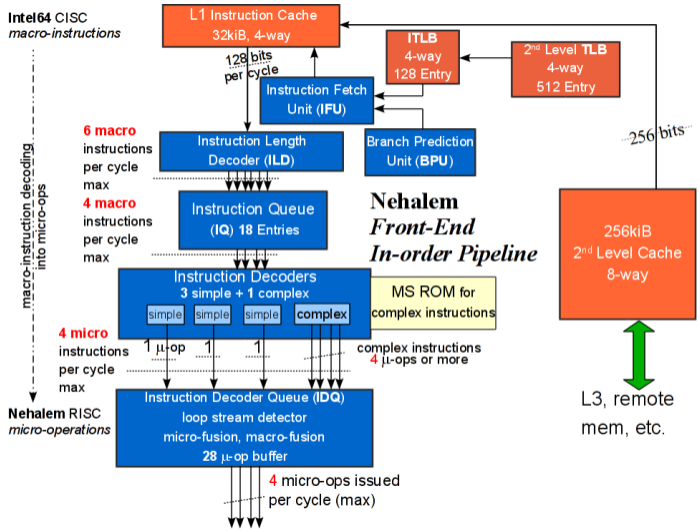
Core i7 Pipeline: Big Picture



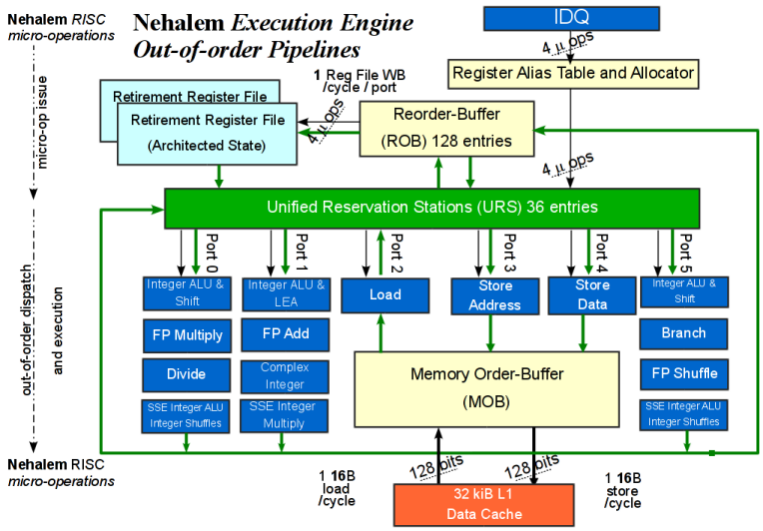
(Images courtesy of Michael E. Thomadakis, Texas A&M Supercomputing Facility)

122

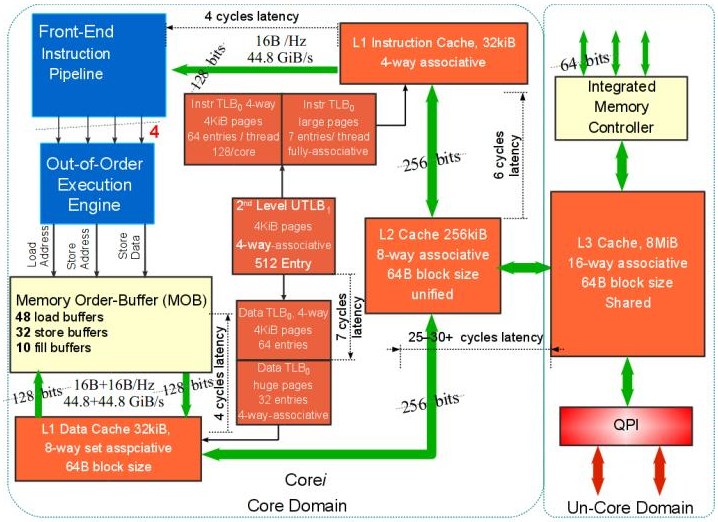
Core i7 Pipeline: Front End



Core i7 Pipeline: Execution Unit

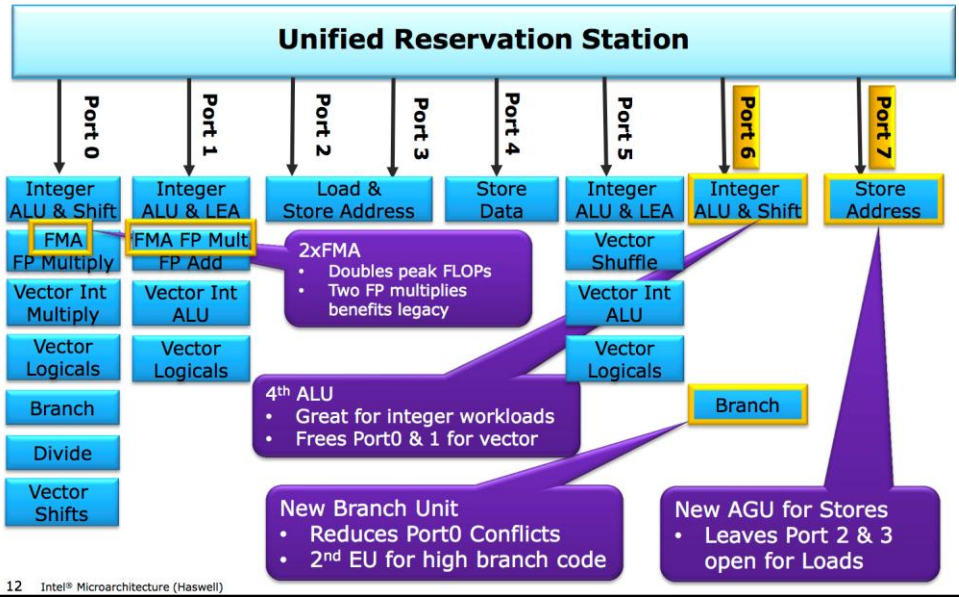


Core i7 Pipeline: Memory Hierarchy



125

Haswell Execution Unit Overview



12 Intel® Microarchitecture (Haswell)

126

Haswell Buffer Sizes

Extract more parallelism in every generation

	Nehalem	Sandy Bridge	Haswell
Out-of-order Window	128	168	192
In-flight Loads	48	64	72
In-flight Stores	32	36	42
Scheduler Entries	36	54	60
Integer Register File	N/A	160	168
FP Register File	N/A	144	168
Allocation Queue	28/thread	28/thread	56

IDF2012 INTEL DEVELOPER FORUM

127

11 Intel® Microarchitecture (Haswell); Intel® Microarchitecture (Nehalem); Intel® Microarchitecture (Sandy Bridge)