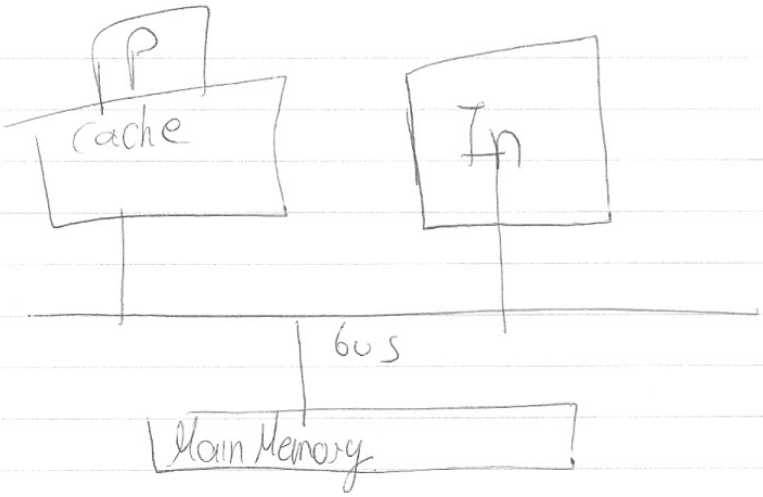
Σοφία Καμπούρη
Α.Μ 3221.

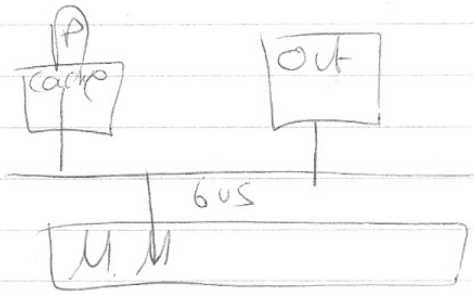
14.1.

α)



Θα διαβάσει αυτά τα νέα δεδομένα αν υπήρχε από τις θέσεις που αντιστοιχούν τα δεδομένα δεν υπάρχει στην κρυφή μνήμη.
Αν υπήρχε τότε ο επεξεργαστής θα πάρει την παλιά τιμή οπότε θα κάνει λάθος

β)

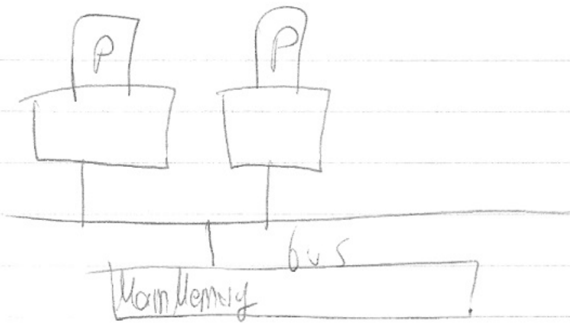


Δεν υπάρχει περίπτωση να φτάσουν λάθος παλιά δεδομένα διότι αν γράφεται στον επεξεργαστή αυτό και στην cache γράφεται και στην Main Memory. Συνεπώς κάθε από τις θέσεις της κρυφής μνήμης αντιστοιχεί στη Main.

γ) Θα καταλήξαν αυτά τα νέα δεδομένα μόνο αν είχε επηρεαστεί η cache memory. Αν δεν έχει επηρεαστεί, η DMA θα διαβάσει μια τιμή απευθείας από τη μνήμη ~~δεδομένα~~ ενώ υπάρχει στην κρυφή και θα είναι λανθασμένη. Η νέα τιμή θα υπάρχει μόνο στην cache.

14.3

sharedVar.



α). Αρχικά η MainMemory έχει την τιμή 0.
Η CPU A στο βήμα ii διαβάζει την μεταβλητή κανονικά (sharedVar = 0) και κάνει υπολογισμούς.
Στο βήμα iii) η CPU B θα διαβάσει sharedVar το νέο sharedVar που υπολόγισε η CPU A και θα το γράψει σε νέα θέση μνήμης cache
Στο βήμα iv) που γίνεται η εκχώρηση του sharedVar = 11 στην CPU A σημαίνει ότι η MainMemory πλέον θα γνωρίζει ότι sharedVar ισούται με 11 αλλά η CPU B θα έχει την παλιά τιμή. Για αυτό το λόγο στο τελευταίο βήμα που η CPU B θα ξαναδιαβάσει την SharedVar θα κάνει υπολογισμούς με τη μη-ενημερωμένη τιμή της.
Αν ήταν write-through τότε κάθε φορά που θα γινόταν αλλαγή στην τιμή της sharedVar θα ενημερωνόταν και η main memory. Συνεπώς η CPU B δεν θα είχε πρόβλημα στο να διαβάζει και να κάνει υπολογισμούς με την νέα τιμή.

β) Αφού η CPU B διαβάσει την sharedVar από την CPU A θα αλλάξει η CPU A σε κατάσταση Shared και όχι σε M γιατί δεν θα είναι η μοναδική μνήμη με το μοναδικό αντίγραφο. Άρα όταν στο βήμα iv η CPU εκτελεί το sharedVar = 11 θα ενημερώσει το bus και θα αλλάξει και η τιμή του sharedVar στην CPU B άρα στο βήμα v θα έχει την σωστή τιμή για τους υπολογισμούς. Μετά θα μπει σε κατάσταση Modified.

γ) Στην περιπτωση MSI μετα το βημα IV γνωριζω οτι καμια αλλη κρυφη μνημη δεν εχει κρυφη μνημη αρα η CPU A βρισκεται σε κατασταση S και οταν η CPU B θελει να διαβασει της sharedVar θα διαβασει την αρχικη τιμη του sharedVar = 0

Στην περιπτωση MESI εχουμε μια παρομοια περιπτωση και η κατασταση S δηλωνει οτι υπαρχει ενδεχομενο να θελει και καποια αλλη κρυφη μνημη να διαβασει τα περιεχομενα της sharedVar αρα η τιμη της θα ενημερωθει και οταν η CPU B θα διαβαζει sharedVar = 11.

14.4

a)
```
lw $t0, 120($0)
lw $t1, 124($0)
lw $t2, 128($0)
lw $t3, 132($0)
addi $t2, $t0, 2
addi $t3, $t1, -1
```

b)
```
lw $t0, 120($0)
lw $t2, 128($0)
addi $t2, $t0, 2
lw $t3, 132($0)
lw $t1, 124($0)
addi $t3, $t3, -1.
```

Επιτρεπεται η αναδιαταξη διοτι ο $t1 δεν χρειαζεται να γινει load πριν χρησιμοποιηθει οποτε ο compiler για να μη χασει κυκλους ρολογιου κανει αναδιαταξη.

γ) εγγραφη στην θεση μνημης που βρισκεται ο ptr.

Δεν μπορει να κανει instruction scheduling διοτι δεν υπαρχει καποια καθυστερηση στους κυκλους ρολογιου

Αν ο $14 και $15 εχουν την ιδια τιμη τοτε θα γινεται εγγραφη και αλλαγη στην ιδια θεση μνημη αρα θα υπαρχει καθυστερηση

ρτ = 14, ρδ = 13

```
lw $14 0($12)
addi $14, $0, 2
sw $14, 128($0)
lw $15, 0($13)
addi $15, $0, -1
sw $15, 132($0)
```

η εντολή δεν υποχρεούται να περιμένει στον ρτ και ρδ έχουν την ίδια τιμή περιμένει να γίνει το store στην μνήμη

19.5

α) 80 - 8 = 72 κύκλοι ρολογιού

γ) 72 από τις 80 εντολές του προγράμματος Β θα εκτελεστούν στην αστοχία της load του προγράμματος Α. Οι υπόλοιποι 8 θα εκτελεστούν μετά την εκτέλεση του προγράμματος Α

β) ~~Επ~~ ~~τρέχει πιο γρήγορα~~ Στην αμυγκολλητή περίπτωση έχουμε χονδρή πολυνημάτωση που σημαίνει ότι υπάρχει περιορισμένη ικανότητα αναπαύσης μετά από την ~~[illegible]~~ αστοχία μνήμης. Επειδή η διοχέτευση παγώνει χάνονται κύκλοι ρολογιού για να "γεμίσει" η διοχέτευση. Άρα τρέχει πιο γρήγορα διότι η επαναπλήρωση της διοχέτευσης γίνεται πιο γρήγορα από ότι η καθυστέρηση γενικά.