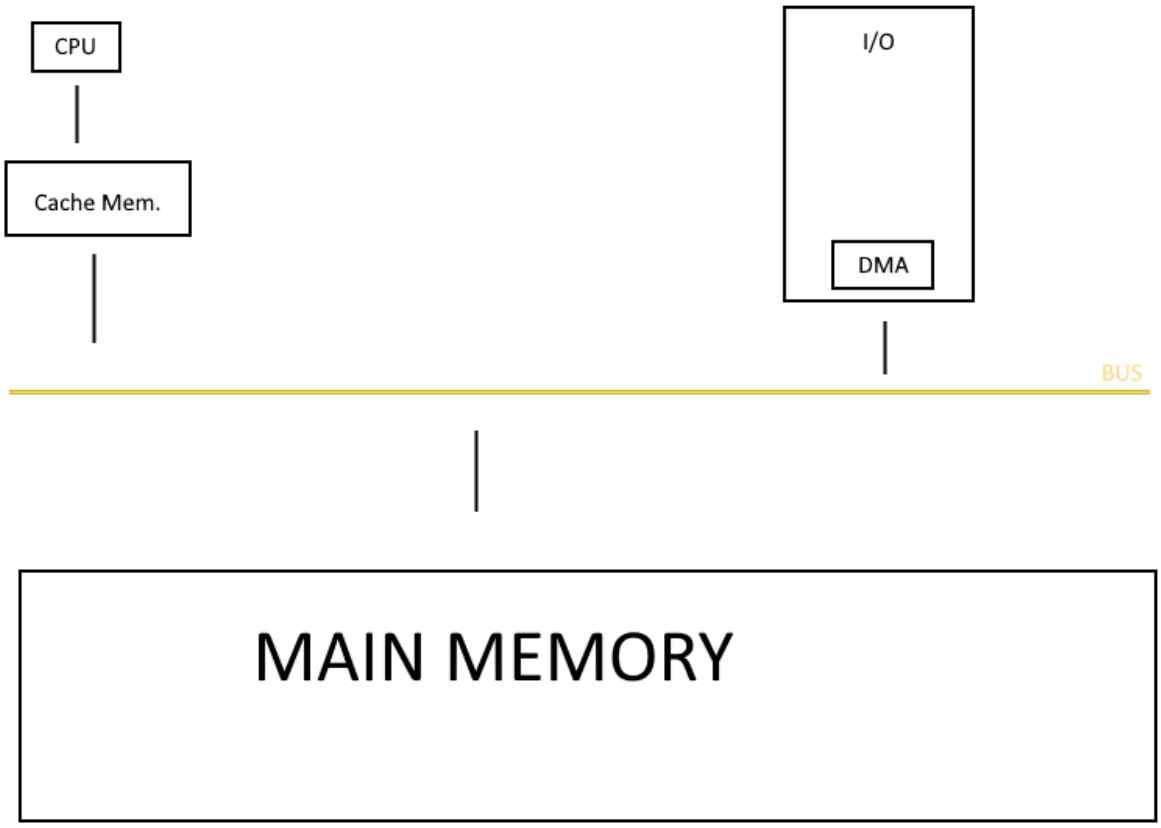
Σειρά Ασκήσεων 14

HY225 Mike Bastakis csd4406

14.1)



a)

Ο επεξεργαστής θα διαβάσει τα σωστά δεδομένα μόνον αν δεν υπήρχε ήδη πριν γίνει το Input η διεύθυνση των δεδομένων στην cache.Δηλαδή αν πριν γίνει το Input από το DMA τα δεδομένα είχαν κάποιες παλιές τιμές και ήταν μέσα στην cache ο επεξεργαστής θα κάνει load τις παλιές τιμές και αρά θα πάρει λάθος δεδομένα από το main memory.

b)

Εφόσον τώρα η κρυφή μνήμη του επεξεργαστή είναι write-through αυτό σημαίνει ότι όταν γίνουν store τα δεδομένα κατευθείαν εγγράφονται και στην cache και στην main memory οπότε δεν υπάρχει περίπτωση στο bus να υπάρχει παλιό δεδομένο και να το πάρει η Output Device.

c)

Τώρα έχοντας cache write-back η οποία στέλνει τιμές στην main memory μόνο όταν είναι έτοιμες να αλλάξουν μπορούμε να στείλουμε παλιά δεδομένα στο output device.Όταν το output device ζητήσει ένα δεδομένο το οποίο βρίσκεται στην cache αλλά δεν έχει αλλάξει ακόμα, αρά δεν βρίσκεται στην main memory θα πάρουμε το παλιό δεδομένο που βρίσκεται στην main memory.Αν όμως το output device ζητήσει δεδομένο το οποίο έχει αλλάξει και δεν βρίσκεται πλέον στην cache αλλά στην main memory τότε θα πάρουμε το σωστό δεδομένο.

14.3)

A)

Αρχικά ο επεξεργαστής A ψάχνει την κοινόχρηστη μεταβλητή στην cache αλλά εφόσον δεν υπάρχει θα ψάξει στην main memory.Όταν την πάρει και φτάσει η μεταβλητή αυτή την cache, αν βρίσκεται στην cache κάποιο άλλο block στην θέση που θα μπει τότε διώχνει το block από την cache και τοποθετεί το block της κοινόχρηστης μεταβλητής. Στην συνέχεια το ίδιο ακριβώς θα κάνει και ο επεξεργαστής B. Ο A αλλάζει την ShVar σε 1 και επειδή ο A έχει write-back cache δεν θα γίνουν αλλαγές στο main memory μέχρι να ξανά αλλάξει ο επεξεργαστής την τιμή της. Ο B πάλι θα διαβάσει την μεταβλητή που βλέπει από την cache του και θα είναι 0 η τιμή της ακόμα και αν την διάβαζε από το main memory πάλι 0 θα ήταν καθώς η cache του A είναιwrite-back και δεν έχει αλλάξει ακόμα την τιμή αφού δεν έχει έρθει κάποιο καινούργιο block.

Προαιρετικό: (Στην περίπτωση που η cache του A ήταν write-through θα άλλαζε μόνο το όταν ο A άλλαζε την τιμή της ShVar = 1 τότε αυτό θα γραφόταν στην main memory ταυτόχρονα όπως και στην cache του. Οπότε όταν το B θα ξανά κοιτούσε την τιμή του ShVar αν είχε διώξει την μεταβλητή από την cache του τότε θα έβλεπε ότι ήταν 1 διαφορετικά αν η μεταβλητή υπήρχε στο cache ακόμα τότε πάλι ο B θα έβλεπε ότι η τιμή είναι 0.)

B)

Αρχικά ο επεξεργαστής A αφού δεν βρίσκει την μεταβλητή στην cache του την αναζητάει στο bus οπού και του την στέλνει η main memory. Αφού επίσης έχουμε μόνο καταστάσεις MSI η μεταβλητή εφόσον δεν μπορούμε να την αφήσουμε χωρίς κατάσταση και είναι clean της βάζουμε την κατάσταση S(shared). Μετά όταν ο B θα ζητήσει την μεταβλητή και δεν θα την βρει στο cache του θα την ζητήσει από το bus οπού και θα την πάρει από την cache του A (αφού έχουμε πρωτόκολλο snooping η cache του A απαντάει πρώτη από την main memory), και θα δώσει ο B την κατάσταση S(shared) όπως και ο A. Ο A αλλάζει την τιμή της μεταβλητής και επειδή έχει κατάσταση S(shared) η Cache θα διαδώσει στο bus ότι η μεταβλητή άλλαξε και θα πει στις υπόλοιπες cache να κάνουν invalidate το αντίγραφο της μνήμης. Και όπως είναι φυσικό η κατάσταση της μεταβλητής θα αλλάξει από S σε M(modified). Τέλος όταν το B δει ότι η μεταβλητή έχει Invalid κατάσταση θα την ζητήσει από το bus όπου και θα απαντήσει πάλι η cache του A και θα δώσει την νέα τιμή της μεταβλητής και θα κάνει την κατάσταση της μεταβλητής στη cache του B S(shared).

Προαιρετικό: (Αν υπήρχε κατάσταση Exclusive : Η μεταβλητή την πρώτη φορά που θα την έπαιρνε ο A από το main memory θα της έβαζε κατάσταση E(Exclusive), Και όταν θα την έδινε στο B όταν την ζητούσε για πρώτη φορά ο B θα άλλαζε την κατάσταση σε S(shared)).

Γ)

Η διαφορά στις δυο αυτές περιπτώσεις όπως προ αναφέρθηκε θα ήταν όταν στην (i) το Α θα έπαιρνε την μεταβλητή από το main memory θα την έβαζε σε κατάσταση Ε αν (MESI) ενώ αν (MSI) θα την έβαζε σε κατάσταση S.

14.4)

Α)

```
1 lw x5, 120(x0)
2 addi x5,x5,1
3 sw x5,128(x0)
4 lw x5,124(x0)
5 addi x5,x5,2
6 sw x5,132(x0)
```

Β)

Έχοντας pipeline 5 βαθμίδων παρατηρούμε ότι το πρόγραμμα θα χάσει 2 κύκλους ρολογιού εφόσον κάνουμε load και αμέσως μετά προσπαθούμε να κάνουμε πράξεις με την μεταβλητή που κάναμε load όμως η μεταβλητή δεν θα έχει διαβαστεί ακόμα την τιμή από το MEM οπότε για κάθε load χάνουμε 1 κύκλο και έχουμε 2 load(επίσης επειδή χρησιμοποιούμε την ιδιά μεταβλητή έχουμε εξαρτημένες εντολές και γιαυτο χάνουμε και τον δεύτερο κύκλο.). **Συνολικά 12 cycles και CPI 2**.

Με instruction scheduling έχουμε :

```
1 lw x5, 120(x0)
2 lw x6,124(x0)
3 addi x5,x5,1
4 sw x5,128(x0)
5 addi x5,x5,2
6 sw x5,132(x0)
```

Τώρα δεν χάνουμε κύκλο γιατί κάνουμε τα load και προλαβαίνουν να πάρουν την τιμή και επίσης δεν χρησιμοποιούμε την ιδιά μεταβλητή για load. **Συνολικά 10 cycles και CPI 1.67**.

Γ)

```
1 lw x5, 0(x14)
2 addi x5,x5,1
3 sw x5,0(x16)
4 lw x5, 0(x15)
5 addi x5,x5,2
6 sw x5,0(x17)
```

Για να κάνει ο compiler instruction scheduling πρέπει να γνωρίζει αν το px,pb κοιτάνε στην ίδια θέση μνήμης. Καθώς αν κοιτάνε στην ιδιά τότε σε αυτήν την θέση θα προσθέσουμε +1 την πρώτη φορά και +2 την δεύτερη οπότε θα έχουμε σε μια θέση μνήμης +3. Αυτό συμβαίνει για στο px βάζουμε την τιμή (*pa)+1 το οποίο θα τον κάνει να κοιτάει στο pb άρα όταν θα κάνουμε py = pb + 2 => py = pa + 1 + 2. Αν όμως οι τιμές του pb px δεν είναι ιδίες τότε μπορεί να κάνει instruction scheduling όπως κάναμε και στο πάνω ερώτημα χρησιμοποιώντας δυο διαφορετικούς καταχώρητες για load. Όμως επειδή ο compiler

δεν ξέρει την τιμή που θα έχουν οι pointers δεν μπορεί να γίνει σε αυτήν την περίπτωση instruction scheduling.

Δ)

Η δεύτερη load αναγκάζεται να περιμένει μόνο αν οι px και pb έχουν ίδια τιμή αφού πρέπει να περιμένει να πάρει τα νέα δεδομένα δηλαδή να έχει τελειώσει η store του προηγούμενου. Σε όλες τις άλλες περιπτώσεις δεν χρειάζεται να περιμένει αφού η δεύτερη load δεν εξαρτάται από την προηγουμένη store. Το hardware μπορεί να κάνει instruction scheduling εφόσον μπορεί να ξέρει την τιμή τον pointers σε αντίθεση με τον compiler. Οπότε καταλήγουμε ότι στο 1% των περιπτώσεων όπου px == pb χάνεται χρόνος ,ενώ στο 99% των φορών όταν δηλαδή px != pb δεν χάνεται.

14.5)

Α)

Αφού ο επεξεργαστής δεν έχει multithreading τότε άμα γίνει αστοχία στην load εξαιτίας του scheduling που κερδίζουμε 8 κύκλους θα χάναμε συνολικά 80-8 = 72 κύκλους.

Β)

Όταν δεν είχε multithreading έχασε 72 κύκλους το thread Α τώρα που έχει το thread Α αφού η προσοχή στρέφεται προς το Β χάνει και τους 80 κύκλους οπότε με Multithreading το Α είναι πιο αργό.

Γ)

Όμως συνολικά το πρόγραμμα με multithreading δεν θα χάσει κύκλο ρολογιού αφού όσο το Α θα περιμένει και δεν θα κάνει κάτι χρήσιμο το Β θα τρέχει κανονικά αφού το Β είναι τυχερό και για 80 κύκλους δεν θα του συμβεί και αυτού κάποια αστοχία.