Η Αρχιτεκτονική του Σετ Εντολών

Instruction Set Architecture

1

Ατζέντα

- Συμβιβασμοί σχεδίασης της αρχιτεκτονικής Σετ Εντολών (ISA)
  - X86
  - RISC vs CISC

2

Κατάσταση του Επεξεργαστή και Δεδομένα

- Η πληροφορία που αποθηκεύεται στον επεξεργαστή στο τέλος της εκτέλεσης μιας εντολής ώστε να βοηθήσει να υπάρχει ένα περιβάλλον για την εκτέλεση της επόμενης εντολής

  Συσσωρευτής, καταχωρητής, μετρητής προγράμματος....

- Αν υπάρχει η δυνατότητα διακοπής της εκτέλεσης μιας εντολής τότε το hardware πρέπει να αποθηκεύει και να επαναφέρει την κατάσταση της εκτέλεσης

  Το υπολογιστικό μοντέλο για τον προγραμματιστή είναι μια σύμβαση/συμβόλαιο μεταξύ του υλικού και του λογισμικού

3

Σετ Εντολών Επεξεργαστή

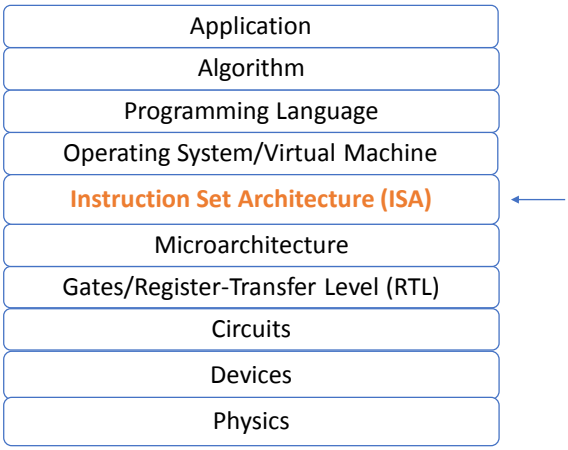
Η Αρχιτεκτονική ενός Σετ Εντολών ή ISA καθορίζει τις διαθέσιμες εντολές για να ελέγχεται η αλλαγή – επεξεργασία της πληροφορίας που υπάρχει μέσα στον επεξεργαστή

- Ένα σετ εντολών ενός επεξεργαστή καθορίζει
  - Έναν τρόπο προσπέλασης των καταχωρητών και της μνήμης
  - Τους υπολογισμούς (πράξεις) που μπορεί να εκτελέσει
  - Έναν τρόπο ελέγχου της ακολουθίας των εντολών για εκτέλεση
  - Την αναπαράσταση (κωδικοποίηση) όλων των παραπάνω στο δυαδικό

Η Αρχιτεκτονική ενός Σετ Εντολών ή ISA πρέπει να ικανοποιεί τις ανάγκες του λογισμικού: assembler, compiler, OS, VM, …

4

Αφαιρετικότητα στα Μοντέρνα Συστήματα



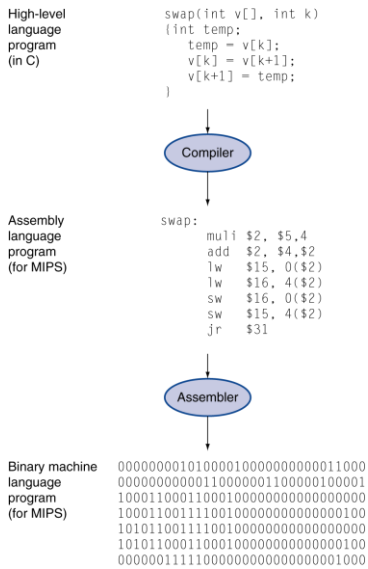
ISA ορίζει τη διασύνδεση – τις συμβάσεις - **(functional contract)** μεταξύ hardware και software

Αυτό σημαίνει, τι εγγυάται το hardware και τι δεν εγγυάται ότι κάνει

5

Αφαιρετικότητα και Πρόγραμμα

- Γλώσσα Προγραμματισμού Ψηλού Επιπέδου
  - Αφαιρετικότητα πιο κοντά στην περιοχή του προβλήματος που επιχειρεί να λύσει
  - Στόχος είναι η παραγωγικότητα και η συμβατότητα
- Γλώσσα σε Assembly
  - Αναπαράσταση του ISA σε μορφή κειμένου
- Αναπαράσταση Υλικού
  - Δυαδική αναπαράσταση των εντολών του ISA



Μικρο-αρχιτεκτονική

6

Αρχιτεκτονική Σετ Εντολών vs Υλοποίηση

- Η Αρχιτεκτονική Σετ Εντολών ορίζει την διεπαφή μεταξύ υλικού και λογισμικού
- Η Αρχιτεκτονική Σετ Εντολών είναι ένα επίπεδο αφαίρεσης που καλύπτει, κρύβει την υλοποίηση από το λογισμικό
- Η ISA ορίζει:
  - Τι λειτουργίες μπορούν να γίνουν (operations, sequencing)
  - Πως ονομάζονται οι θέσεις αποθήκευσης, ποια κατάσταση του επεξεργαστή είναι ορατή από τον προγραμματιστή
  - Το format (bit encoding) των εντολών
- Συνήθως η ISA δεν ορίζει:
  - Timing of the operations
  - Power used by operations
  - How operations/storage are implemented

7

Στόχοι της Αρχιτεκτονικής Σετ Εντολών

- Ευκολία προγραμματισμού / δημιουργίας κώδικα software
- Ευκολία σχεδίασης, υλοποίησης hardware
- Καλές επιδόσεις
- Συμβατότητα

- Συμπαγής – ελαχιστοποίηση μεγέθους κώδικα
- Επεκτασιμότητα, βελτιστοποίηση
- Χαρακτηριστικά υποστήριξης OS, παραλληλισμού, ...

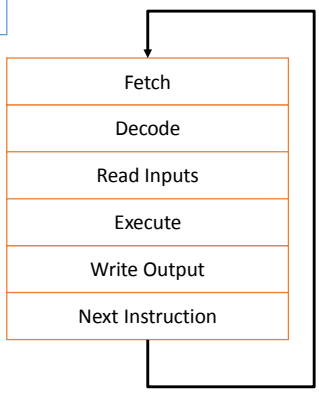
8

Υλοποίηση ISA

Η Αρχιτεκτονική του Σετ Εντολών θα πρέπει να είναι ανεξάρτητη από τις μεθόδους σχεδίασης και βελτιστοποίησης της υλοποίησης με στόχο υψηλότερες επιδόσεις

- Παραδείγματα
  - Εντολές μεταβλητού μήκους
  - Ποικιλία φορμάτ εντολών
  - Έμμεσοι καταχωρητές
  - Πολύπλοκοι τρόποι διευθυνσιοδότησης
  - Interrupts ακριβείας
  - Εκτέλεση εντολών κατά ατομικό τρόπο

Ένα απλό pipeline επεξεργαστή

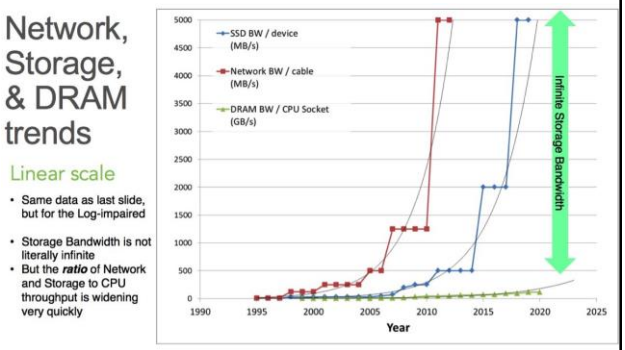


Όμως..... Τι σημαίνει επιδόσεις ?

9

Επιδόσεις Επεξεργαστή

- Χρόνος απόκρισης, καθυστέρηση (response time, latency)
  - Χρόνος που απαιτήθηκε από την αρχή ως το τέλος συμπεριλαμβάνοντας όλα
  - Χρόνος στον επεξεργαστή (CPU time) (μόνο εντός του επεξεργαστή)
- Ρυθμός/εύρος εκτέλεσης (throughput / bandwidth)
  - Πόση δουλειά μπορεί να γίνει στη μονάδα του χρόνου



10

Επιδόσεις - Καθυστέρηση

- Χρόνος απόκρισης (Elapsed time)
  - Χρόνος που απαιτήθηκε από την αρχή ως το τέλος συμπεριλαμβάνοντας όλα τις καθυστερήσεις
- Χρόνος στον επεξεργαστή (CPU time) (μόνο εντός του επεξεργαστή)

$$CPU\ Time = CPU\ clock\ cycles \times CPU\ clock\ time$$

- CPU clock cycles = αριθμός κύκλων ρολογιού που απαιτήθηκαν για την εκτέλεση του προγράμματος
  - # of instructions executed
  - Cycles per instruction
- CPU clock time = 1 / frequency

11

Ο Νόμος της Επίδοσης

- $CPU\ Time = \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$
- Τι επηρεάζει κάθε όρο ? Πως επηρεάζει η ISA κάθε όρο?
- Instructions / program = αριθμός εντολών που **εκτελούνται** (ΟΧΙ μέγεθος κώδικα)
  - Καθορίζονται από το συγκεκριμένο πρόγραμμα, τον compiler, και ISA
- Cycles / instruction (CPI)
  - Καθορίζονται από την ISA, μarch, program, και τον compiler
- Seconds / cycle (critical path)
  - Καθορίζονται από την μarch και την τεχνολογία

12

Κύκλοι ρολογιού ανά Εντολή (CPI)

- Κάθε κατηγορία εντολών απαιτεί διαφορετικό αριθμό κύκλων
- Στην πραγματικότητα, ακόμα και στην ίδια κατηγορία κάθε/κάποια εντολή μπορεί να απαιτεί διαφορετικό αριθμό κύκλων

- Στην πραγματικότητα εννοούμε: ***Weighted CPI***

$$CPI = \frac{Clock\ cycles}{Instruction\ count} = \sum_{i=1}^{n} CPI_i \times \frac{Instruction\ count_i}{Instruction\ count}$$

13

Χρόνος Επεξεργαστή

- $CPU\ Time = \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$

- Πως βελτιώνεται ο χρόνος ?
  - Ελάττωση αριθμού εντολών
  - Ελάττωση αριθμού κύκλων ανά εντολή
  - Ελάττωση χρόνου κάθε κύκλου ρολογιού

- Αλληλοεξαρτήσεις, αλληλοεπιδράσεις ?

14

Παράδειγμα CPI – καλύτερη μηχανή ?

- Computer A: Cycle time = 250ps, CPI = 2.0
- Computer B: Cycle time = 500ps, CPI = 1.2

Ίδια αρχιτεκτονική σετ εντολών (ISA)

$$CPU\ time_A = Instruction\ count \times CPI_A \times Cycle\ time_A = Instruction\ count \times 2 \times 250ps = Instruction\ count \times 500ps$$

A is faster…

$$CPU\ time_b = Instruction\ count \times CPI_B \times Cycle\ time_B = Instruction\ count \times 1.2 \times 500ps = Instruction\ count \times 600ps$$

$$\frac{CPU\ time_B}{CPU\ time_A} = \frac{600ps}{500ps} = 1.2$$

…by this much

15

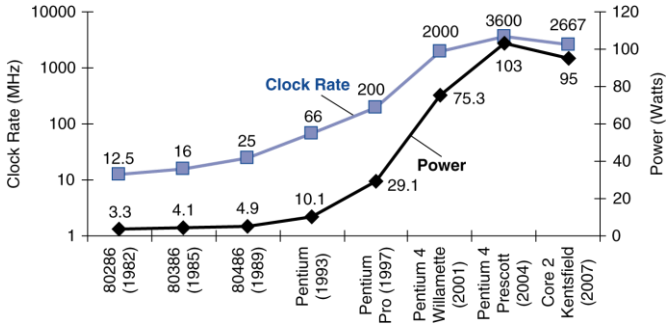
Άλλες Μετρικές “Επιδόσεων”

- Οι επιδόσεις δεν περιορίζονται στην καθυστέρηση/απόκριση του επεξεργαστή

- Κατανάλωση ισχύος
- Χώρος (σε mm^2 Si, a.k.a., # transistors)
- Πολυπλοκότητα
- Επεκτασιμότητα
- Συμβατότητα

16

Κατανάλωση Ισχύος και Τεχνολογία CMOS



Εξέλιξη της τεχνολογίας ολοκληρωμένων σε CMOS:

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30 | 5V → 1V | ×1000

17

Συμβατότητα

- Ο στόχος της αρχιτεκτονική ISA είναι να διαχωρίσει το interface από την υλοποίηση
- Πολλές διαφορετικές υλοποιήσεις
  - IBM/360: σχεδίασε 7 διαφορετικές μηχανές όλες με το ίδιο ISA
  - Intel: 8086 → core i7 → Xeon Phi → ?
  - ARM ISA: mobile → server
- **Προστασία της επένδυσης σε λογισμικό**

- Είναι σημαντική απόφαση τι πρέπει να "φαίνεται" και τι να είναι κρυμμένο.
  - E.g., MIPS "branch delay slots"

18

Τι Πρέπει να Καθορίζει η ISA ?

- Τελεστές
  - Αριθμός τελεστών ανά εντολή?
  - Τι είδους?
  - Μηχανισμοί προσπέλασης
- Πράξεις
  - Τι είδους?
  - Πόσες διαφορετικές?
- Format/κωδικοποίηση
  - Μήκος της ακολουθίας δυαδικών ψηφίων
  - Ορισμός στην ακολουθία δυαδικών ψηφίων ποια bits κωδικοποιούν τι

19

Ορισμός Τελεστών ↔Μοντέλο Μηχανής

- Τρεις βασικοί τύποι μηχανής
  - Stack
  - Accumulator
  - Register
- Δύο τύποι μηχανών με καταχωρητές
  - Καταχωρητές-μνήμη: οι τελεστές μπορεί να είναι είτε καταχωρητές είτε μνήμη
  - Load-store

20

Τελεστές Εντολών

- Stack

0 address	add	push(pop() + pop())

- Accumulator

1 address	add A	Acc ← Acc + mem[A]

- Register-Memory

2 address	add R1, A	R1 ← R1 + mem[A]
3 address	add R1, R2, A	R1 ← R2 + mem[A]

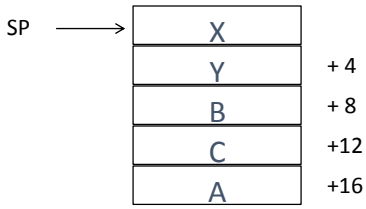
- Load-Store

3 address	add R1, R2, R3	R1 ← R2 + R3
2 address	load R1, R2	R1 ← mem[R2]
	store R1, R2	mem[R1] ← R2

21

Παράδειγμα

- Code for: A=X*Y – B*C



Stack

```
push   8(SP)
push   16(SP)
mult
push   4(sp)
push   12(sp)
mult
sub
st     20(sp)
pop
```

Accumulator

```
ld     8(SP)
mult   12(SP)
st     20(SP)
ld     4(SP)

mult   0(SP)
sub    20(sp)
st     16(sp)
```

reg-mem

```
mult   R1,8(SP),12(SP)

mult   R2,0(SP),4(SP)

sub    16(sp),R2,R1
```

ld/st

```
ld     r1,8(SP)
ld     r2,12(SP)
ld     r3,4(SP)
ld     r4,0(SP)
mult   r5,r1,r2
mult   r6,r3,r4
sub    r7,r6,r5
st     16(SP),r7
```

22

Μοντέλο Μηχανής - Tradeoffs

- Stack and Accumulator:
  - Η κωδικοποίηση των εντολών είναι μικρή
  - Ο αριθμός εντολών είναι μεγάλος (IC)
  - Η αρχιτεκτονική είναι πολύ απλή
- Register-Memory:
  - Η κωδικοποίηση των εντολών είναι πολύ μακρύτερη
  - Περισσότερη “δουλειά” ανά εντολή
  - Ο αριθμός εντολών είναι μικρός (IC)
  - Η αρχιτεκτονική είναι πιο περίπλοκη
- Load/Store:
  - Η κωδικοποίηση των εντολών είναι ενδιάμεση
  - Λιγότερη “δουλειά” ανά εντολή
  - Ο αριθμός εντολών είναι μεγάλος (IC)
  - Η αρχιτεκτονκή είναι πιο περίπλοκη

23

Συνήθεις Τύποι Εντολών

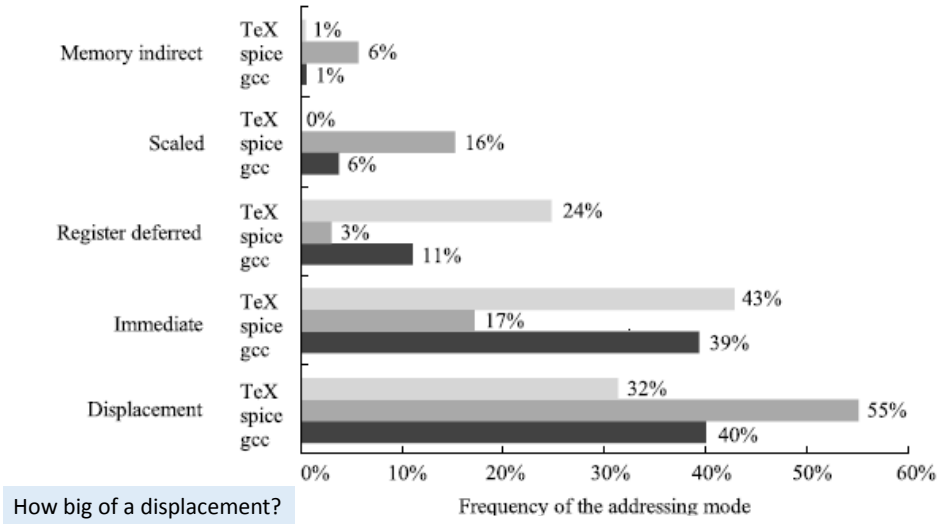
• Register	add r1,r2,r3 add r1,r2
• Immediate	add r1,#7
• Memory	
• direct	add r1,[0x1000]
• register indirect	add r1,(r2)
• displacement	add r1,100(r2)
• indexed	add r1,(r2+r3)
• indexed+displacement	add r1,100(r2+r3)
• scaled+displacement	add r1,100(r2+r3*s)
• memory indirect	add r1,([0x1000])
• autoincrement	add r1,(r2)+
• autodecrement	add r1,(r2)-

24

Τελεστές στην Μνήμη

- Τρόποι (modes) προσπέλασης/διευθυνσιοδότησης της Μνήμης: πως υπολογίζεται η διεύθυνση μνήμης ενός τελεστή?
- Πόσοι διαφορετικοί τρόποι χρειάζονται?
- Πόσο περίπλοκοι τρόποι?
- Πόση μνήμη μπορεί να προσπελαστεί με τον κάθε τρόπο?
- Trade-offs
  - Πόσο χρήσιμος είναι ο κάθε τρόπος?
  - Ποιά είναι η επίδραση του κάθε τρόπου στο CPI? IC? Freq?
  - Πόσα bits απαιτούνται για την κωδικοποίηση του κάθε τρόπου στην εντολή?

Συχνότητα των Τρόπων Προσπέλασης



How big of a displacement?

Πόσοι Καταχωρητές

- Περισσότεροι καταχωρητές σημαίνει:
  - Μεγαλύτερη κωδικοποίηση, μακρύτερες εντολές
  - Η προσπέλαση καθενός καταχωρητή γίνεται πιο αργή and/or
  - Μεγαλύτερη κατανάλωση ισχύος ανά προσπέλαση
  - Περισσότερη πληροφορία κατάστασης είναι ανοιχτή..
    (more saves/restores per func call, context switch, …)
- Λιγότεροι καταχωρητές σημαίνει :
  - Μεγαλύτερη δυσκολία για τον compiler
  - Θεωρείστε τους καταχωρητές ως “cache level-0”
  - Μικρές εντολές
  - Περισσότερες εντολές
- Η Τάση είναι προς περισσότερους καταχωρητές. Γιατί ?

27

Λειτουργίες

- Αριθμητικές
- Λογικές
- Μεταφοράς δεδομένων
- Ελέγχου ροής εκτέλεσης
- Υποστήριξη Λειτουργικού Συστήματος
- Υποστήριξη παραλληλισμού

28