Αρχιτεκτονική Υπολογιστών

Ενότητα 8: Επεξεργαστές CISC και RISC, Είσοδος/Έξοδος, Διαδικασίες, Σωρός.

Δρ. Μηνάς Δασυγένης
mdasyg@ieee.org
Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών
http://arch.ece.uowm.gr/mdasyg

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σκοπός ενότητας

- Η κατανόηση της διαφορετικότητας των CISC και RISC κεντρικών μονάδων επεξεργασίας.
- Η κατανόηση των μηχανισμών εισόδου-εξόδου.
- Η κατανόηση της διαφορετικότητας και της σημαντικότητας των διακοπών και εξαιρέσεων.
- Η κατανόηση της λειτουργίας και της σημαντικότητας του σωρού.
- Η χρήση των συναρτήσεων στην x86.

Επεξεργαστές CISC και RISC

Ως προς τον αριθμό των εντολών ISA, υπάρχουν 2 κατηγορίες (1/2)

- **RISC: Reduced Instruction Set Computer:**
  - Η ISA περιλαμβάνει λίγες και απλές εντολές.
  - Σταθερή κωδικοποίηση εντολών.
  - Ο κώδικας χρησιμοποιεί έναν μεγάλο αριθμό από απλές εντολές που. υλοποιούνται εύκολα στο υλικό και εκτελούνται γρήγορα.
  - Αρχές δεκαετίας 80΄:
    - ✓ MIPS Project, Stanford University, Καθ. John Hennessy.
    - ✓ RISC Project, UC Berkeley University, Καθ. David Patterson.

- **CISC: Complex Instruction Set Computer:**
  - Ο κώδικας που χρησιμοποιεί λιγότερες αλλά περισσότερο σύνθετες. εντολές που απαιτούν αρκετά πολύπλοκη υλοποίηση.
  - Αναπτύχθηκαν πριν τις αρχιτεκτονικές RISC.
    - ✓ Όμως ο όρος CISC προέκυψε μετά τον όρο RISC.

Ως προς τον αριθμό των εντολών ISA, υπάρχουν 2 κατηγορίες (2/2)

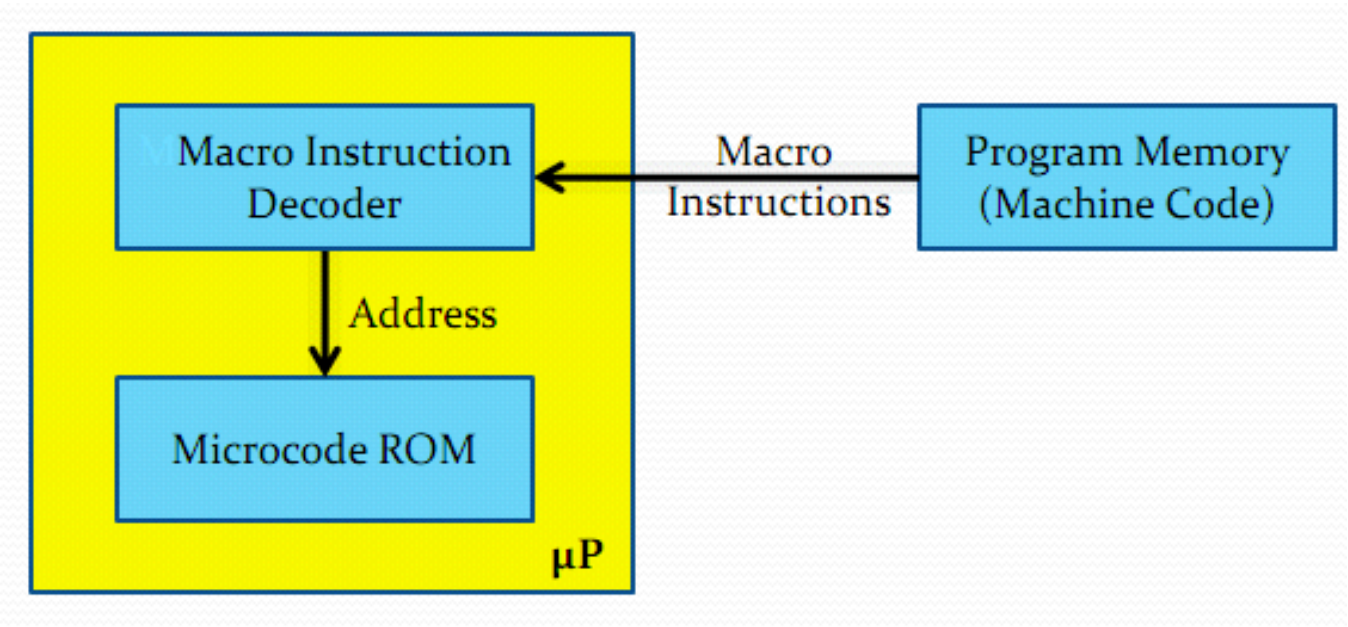
- **RISC: Reduced Instruction Set Computer:**
  - ARM, Sun SPARC, MIPS, IMB Power, Alpha.
  - Κέρδισε τις εντυπώσεις ως προς την τεχνολογική υπεροχή.
- **CISC: Complex Instruction Set Computer:**
  - Intel x86(IA32).
  - Κέρδισε την αγορά των Pcs.
    - ✓ Η συμβατότητα αποδείχτηκε περισσότερο ισχυρή παράμετρος από όσο μπορούσε να φανταστεί κανείς.
      - Εκτός βέβαια από την Intel.
    - ✓ Ο Νόμος του Moore συνέβαλλε σημαντικά.
      - Σήμερα η πολύπλοκη αποκωδικοποίηση των εντολών δεν κοστίζει τόσο, αφού η σχετική επιβάρυνση σε αριθμό transistor είναι μικρή!

MIPS vs Intel x86

- MIPS (RISC ISA):
  - Όραμα μιας μικρής ομάδας.
  - Όλα τα επιμέρους κομμάτια της αρχιτεκτονικής ταιριάζουν μεταξύ τους.
  - Απλότητα στη σχεδίαση.
- INTEL x86(CISC ISA):
  - Αποτελέσματα εργασίας **πολλών ανεξάρτητων ομάδων.**
  - Εξελίχθηκε σε διάστημα 20 ετών.
    - Συνεχής προσθήκη νέων χαρακτηριστικών ISA.
    - Συμβατότητα.
- Γιατί στο μάθημα θα δώσουμε περισσότερο βάρος στην ISA και την οργάνωση MIPS αντί για την αρχιτεκτονική Intel x86;
  - Απλότητα στη σχεδίαση, ευκολία στην κατανόηση.
  - Ευρύτατη χρήση σε ενσωματωμένες εφαρμογές (System-On-Chip, SoC).
  - Οι ενσωματωμένοι υπολογιστές κυριαρχούν έναντι των Pcs.

Οι CISC χρησιμοποιούν τον μικροκώδικα μηχανής

- Χαμηλού επιπέδου εντολές που ελέγχουν ένα CISC μP.
- Σταθερά “ καλωδιωμένος ” κώδικας της ROM.
- Σύνολο μικροεντολών για κάθε μακροεντολή.

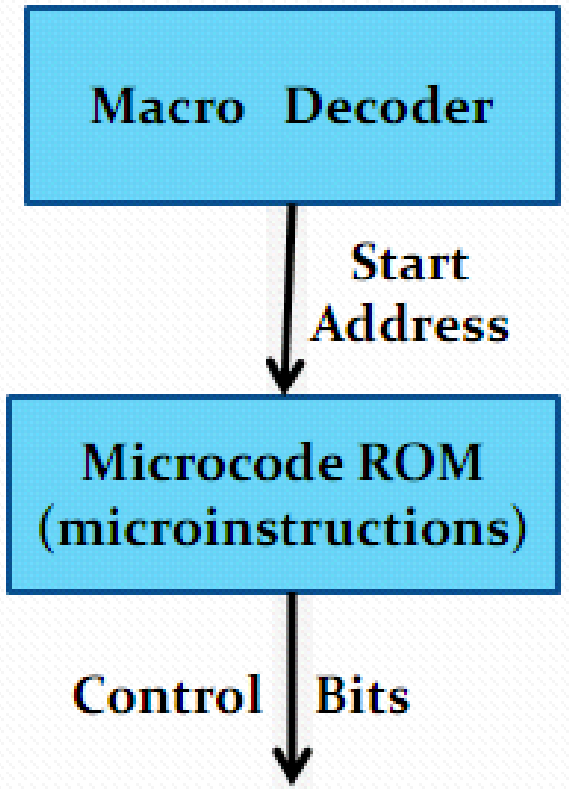


Παράδειγμα micro-ops

- Έστω γράφουμε στην assembly:
  - mov al,buffer[si]
- Αυτή η μακροεντολή μεταφράζεται (μέσα στον επεξεργαστή) στις μικρο-εντολές:
  - load mdr,buffer[si]
  - mov al,mdr
- Έστω γράφουμε στην assembly:
  - add al,buffer[si]
  - Αυτή η μακροεντολή μεταφράζεται:
    - ✓ load mdr,buffer[si]
    - ✓ mov alu_reg1,mdr
    - ✓ mov alu_reg2,al
    - ✓ add
    - ✓ mov al, alu_outputreg

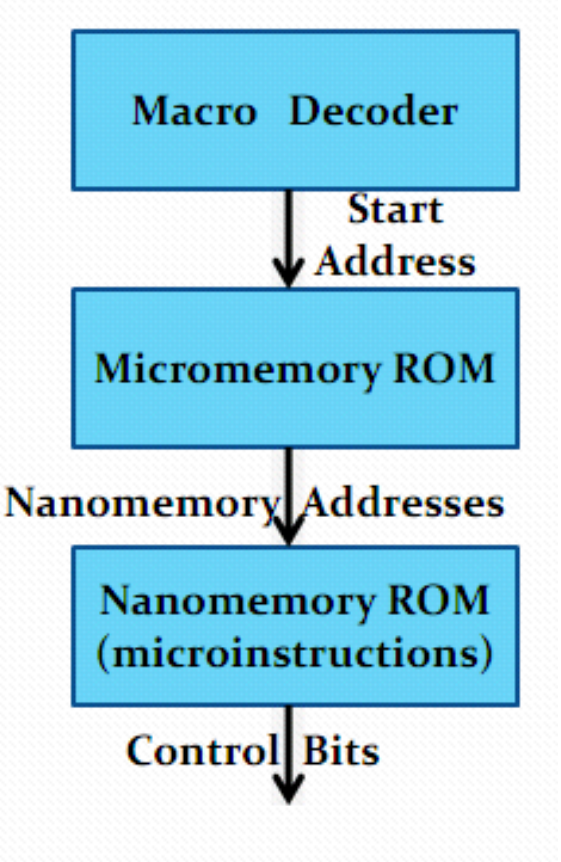
Κάθε εντολή assembly (macro-code) μεταφράζεται σε micro-instructions

- Ένα ενσωματωμένο ROM σε πυρίτιο .
- Συγκεκριμένες μικροεντολές για κάθε μακροεντολή.
- Το μήκος του μικροκώδικα εξαρτάται από την κάθετη ολοκλήρωση των εντολών.
- Κάθετη ολοκλήρωση σημαίνει ότι μερικά κομμάτια ελέγχου εξυπηρετούν περισσότερες από μία λειτουργίες.
- Bits ελέγχου του πολυπλέκτη χρησιμοποιούνται για τον έλεγχο διάφορων λειτουργιών.



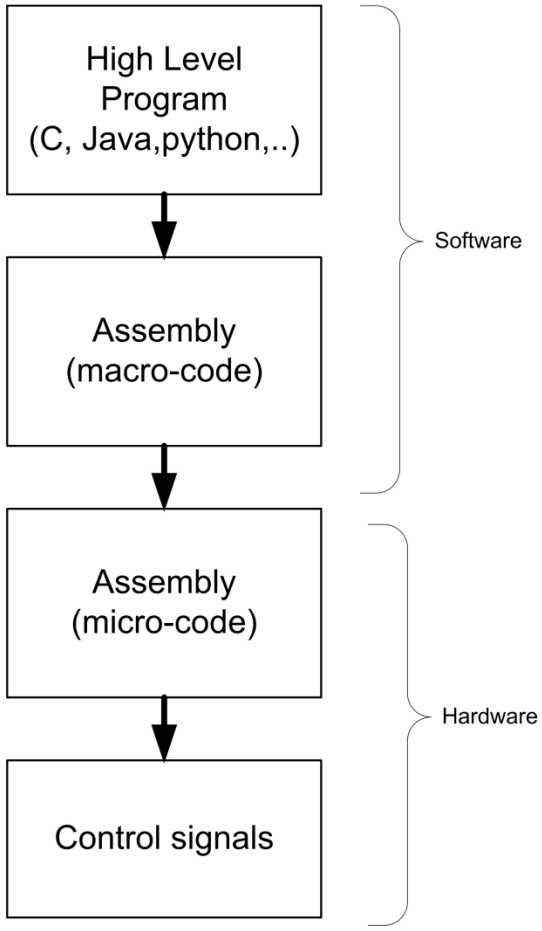
Εναλλακτική υλοποίηση με 2 ROM

- Δύο ενσωματωμένα ROM σε πυρίτιο.
- Αποθήκευση bits μνήμης.
- Επαναχρησιμοποίηση των μικροεντολών της μικρομνήμης της ROM με διάφορες των μακρο εντολών.



Ιεραρχία εντολών & μικροεντολών

- Το πρόγραμμα υψηλού επιπέδου συμβολομεταφράζεται και δημιουργεί κώδικα μηχανής (assembly) από το compiler/assembler.
- Ο κώδικας μηχανής αναλύεται από τη μονάδα ελέγχου σε μικροκώδικα/μικροεντολές και δημιουργούνται τα κατάλληλα σήματα ελέγχου.



Η χρήση του μικροκώδικα

- Η χρήση του μικροκώδικα έχει ένα σημαντικό **πλεονέκτημα**: Απεξαρτητοποιεί τις εντολές του κώδικα μηχανής από το υλικό. Με αυτόν τον τρόπο υπάρχει ευελιξία για την τροποποίηση και τη βελτιστοποίηση των εντολών μηχανής, αλλά και την προσθήκη νέων εντολών.
- Ο μικροκώδικας κάθε συγκεκριμένου επεξεργαστή είναι μοναδικός, ακόμη και σε επεξεργαστές της ίδιας οικογένειας.
- Μερικές φορές ο μικροκώδικας χρησιμοποιείται για να προσομοιώνει λειτουργίες άλλων επεξεργαστών της ίδιας οικογένειας, προκειμένου να υπάρχει συμβατότητα στη assembly (π.χ. προσομοιώνει μια μονάδα πράξεων πραγματικών αριθμών).

Ο μικροκώδικας συμβάλει στη μείωση του κόστους

- Ο μικροκώδικας χρησιμοποιείται για να επιταχύνει το time-to-market, επειδή επιτρέπει να απελευθερωθεί στην αγορά ένας επεξεργαστής χωρίς να έχει δοκιμαστεί σε μεγάλο ποσοστό, αφού αν βρεθεί κάποιο πρόβλημα μπορεί να επιδιορθωθεί με ένα firmware update στο μέλλον (shift the commitment forward).
- Σε διαφορετική περίπτωση θα πρέπει να δοκιμαστεί ενδελεχώς, κάτι που συνεπάγεται μεγάλο κόστος και δε μπορεί να γίνει 100%. Κλασσικό παράδειγμα: το bug του λανθασμένου υπολογισμού της διαίρεσης στον Pentium I. Δεν υποστήριζε μικροκώδικα και έτσι έπρεπε να αποσυρθούν όλοι οι επεξεργαστές και να αντικατασταθούν (με μεγάλο κόστος!).

Ο μικροκώδικας συμβάλει στη βελτίωση των επιδόσεων

- Με τη χρήση μικροκώδικα μπορεί αντί να χρησιμοποιηθεί υλικό (~transistor) για την αντιμετώπιση ειδικών καταστάσεων, να υλοποιηθεί με μικροκώδικα, και το υλικό που θα χρησιμοποιούνταν να έχει μια πιο βέλτιστη χρήση για τις γενικές περιπτώσεις (π.χ. με την τοποθέτηση μιας μεγαλύτερης κρυφής μνήμης).

- Οι σύγχρονες CISC x86 συνήθως αποκωδικοποιούν τις assembly εντολές σε 1-4 εντολές RISC micro-code, κάτι που επιτρέπει την εκτέλεση εκτός σειράς και την αποτελεσματική χρονοδρομολόγηση των μικρο-εντολών.

Οι μικροεντολές σε σύγχρονους επεξεργαστές

- Συνήθως οι εταιρίες κατασκευής επεξεργαστών, δεν αποκαλύπτουν αναλυτικές πληροφορίες για τη λειτουργία των επεξεργαστών τους. Για αυτό το λόγο δεν υπάρχουν αρκετά στοιχεία για την αναλυτική χρήση των μικρο-εντολών.
  - Όμως, η πατέντα της AMD RISC86, παρέχει αρκετές πληροφορίες για το μικροκώδικα του AMD K7: (google.com/patents/US6336178).
- Επίσης, στοιχεία βρίσκονται σε:
  - Bruce Shriver's, "The Anatomy of a High-Performance Microprocessor: A Systems Perspective".
  - realworldtech.com/crusoe-intro/
  - simh.trailing-edge.com/dsarchive.html (non X86 microcode)
  - google.ca/patents/US7464253

Είσοδος / Έξοδος

Η Ε/Ε είναι ένα πολύ σημαντικό τμήμα του υπολογιστή

- Ένα πολύ σημαντικό στοιχείο ενός υπολογιστή είναι το υποσύστημα I/O (Input/Output – Εισόδου/Εξόδου).
- Η Ε/Ε δεν είναι απλά καλώδια και σύνδεσμοι. Είναι λογισμικό + υλικό.
- Τα περιφερειακά Ε/Ε δε συνδέονται απευθείας στο δίαυλο. Απαιτούνται εξειδικευμένα ΟΚ πάνω στον επεξεργαστή ή εκτός για να ελέγχουν την Ε/Ε.

Τρόποι Εισόδου Εξόδου

- Ποιοι μηχανισμοί εισόδου/εξόδου χρησιμοποιούνται στους προσωπικούς υπολογιστές;
  - **Προγραμματισμένη Ε/Ε με αναμονή/απασχόληση (busy waiting).**
  - **Ε/Ε οδηγούμενη από διακοπές (interrupt driven).**
  - **Ε/Ε με άμεση προσπέλαση μνήμης.**

Παράδειγμα:

Ένα τηλέφωνο που το σηκώνουμε κατά διαστήματα και ρωτάμε αν είναι κανείς στην άλλη γραμμή (busy waiting)!

Τι ισχύει για την προγραμματισμένη Ε/Ε με αναμονή/απασχόληση;

- Είναι ο πιο απλός μηχανισμός.
- Ήταν ο πρώτος μηχανισμός Ε/Ε.
- Χαμηλή πολυπλοκότητα υλοποίησης.
- Άμεση απόκριση του επεξεργαστή.
- Μεγάλη επιβάρυνση (απαιτούνται πολλοί κύκλοι). Ο επεξεργαστής ελέγχει συνεχώς αν η Ε/Ε είναι έτοιμη.
- Χρησιμοποιείται σε πολύ απλά συστήματα μικροεπεξεργαστών ή σε ενσωματωμένα.
- Στη x86 υλοποιείται με τις εντολές in και out και χαρτογραφημένη μνήμη Ε/Ε.

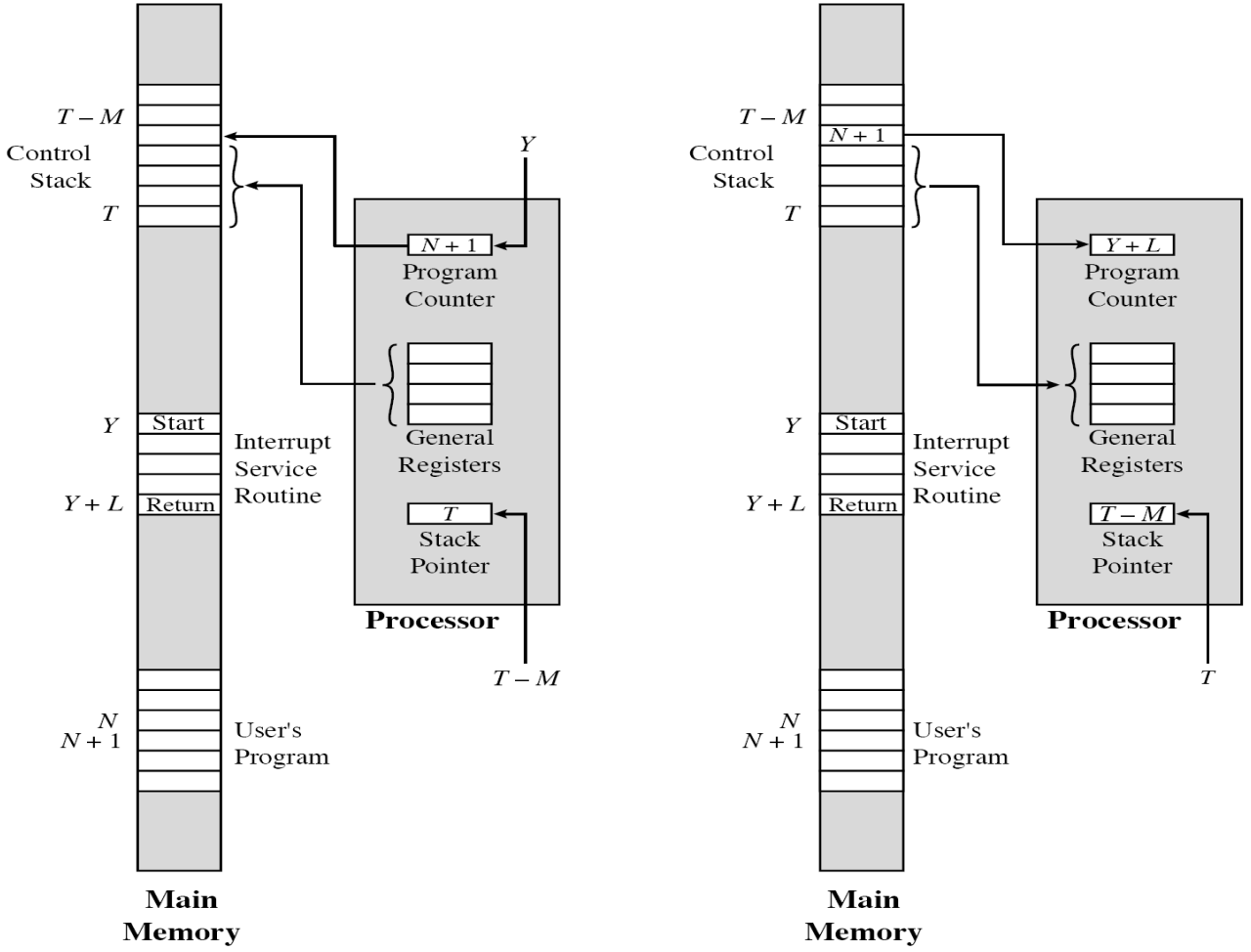
Οι κατηγορίες προγραμματιζόμενου I/O

- Με χαρτογράφηση μνήμης I/O:
  - Ένας κοινός χώρος διευθύνσεων μνήμης για θέσεις μνήμης και συσκευές I/O.
  - Ο επεξεργαστής χειρίζεται τους καταχωρητές δεδομένων I/O ως θέσεις μνήμης.
  - Χρησιμοποιούνται ίδιες εντολές πρόσβασης μνήμης και I/O.
- Με απομόνωση:
  - Ξεχωριστός χώρος διευθύνσεων μνήμης και θέσεις I/O
  - (π.χ. Η θέση μνήμης 0100h είναι διαφορετική από την θέση I/O 0100h).
  - Ξεχωριστές εντολές πρόσβασης μνήμης και I/O.

Η Ε/Ε οδηγούμενη από διακοπές (interrupt driven)

- Ο επεξεργαστής ξεκινά την Ε/Ε και υποδεικνύει στη συσκευή να παράγει μια διακοπή μόλις ολοκληρωθεί.
- Ο επεξεργαστής συνεχίζει τη λειτουργία του κανονικά (δε δαπανάει άλλους κύκλους).
- Η επεξεργασία των διακοπών είναι ακριβή (πολλοί κύκλοι).
- π.χ. Μια διακοπή για κάθε χαρακτήρα που μεταδίδεται.
- Πιο καλή απόδοση από την προγραμματιζόμενη Ε/Ε.

Υλοποίηση διακοπής!



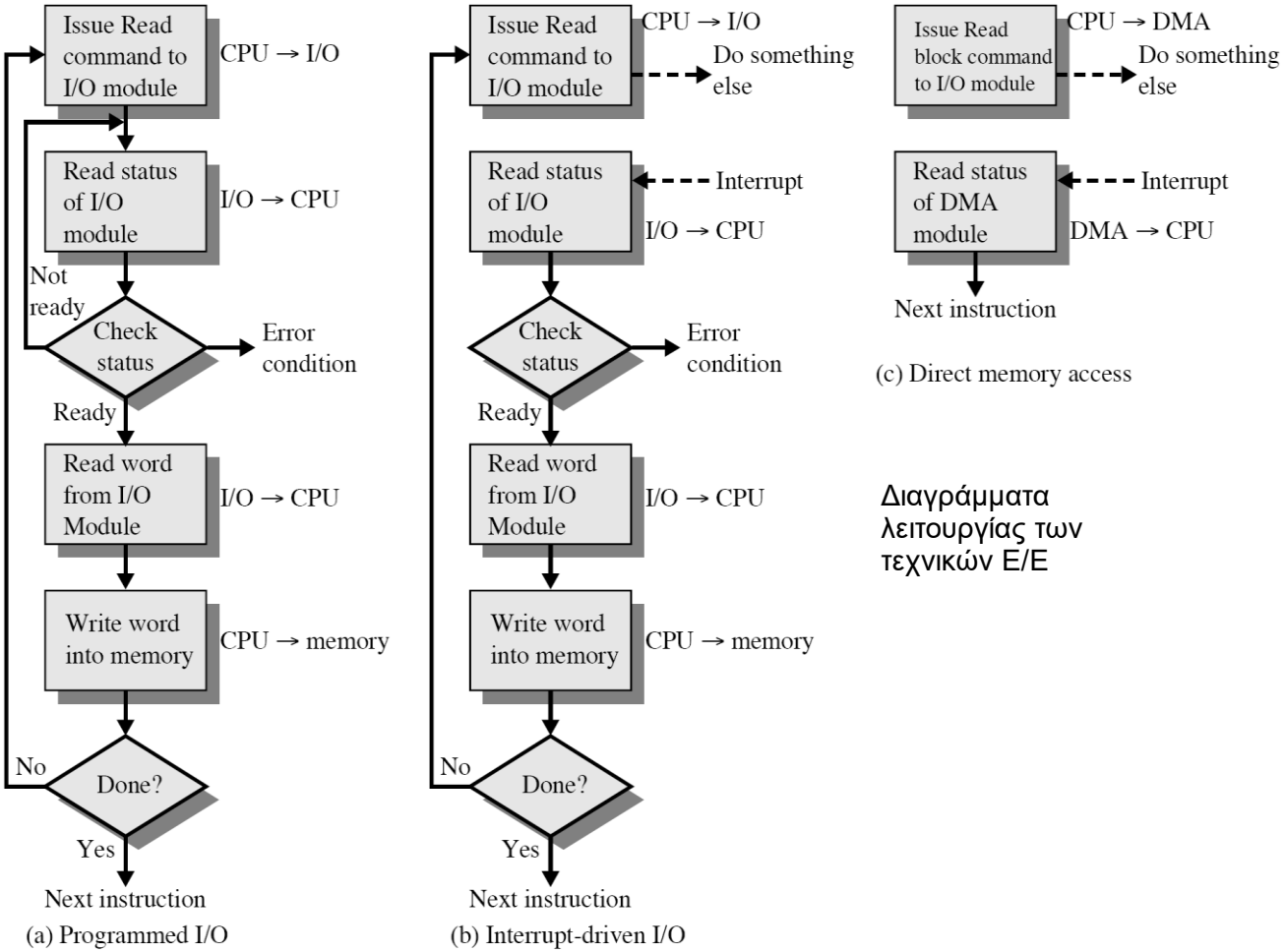
Τι ισχύει για την Ε/Ε με άμεση προσπέλαση μνήμης

- Μοιάζει με την προγραμματισμένη Ε/Ε αλλά το κάνει κάποια άλλη συσκευή- ελεγκτής.
- Υπάρχει μια βοηθητική συσκευή – Ελεγκτής Άμεσης Προσπέλασης Μνήμης (DMA).
- Στην απλή έκδοση αυτή η συσκευή έχει 4 καταχωρητές:
  - Δ/νση εγγραφής ή ανάγνωσης (προέλευση).
  - Αριθμός Bytes.
  - Αρ. συσκευής ή δ/νση μνήμης Ε/Ε που θα χρησιμοποιηθεί.
  - Αν θα διαβαστούν ή εγγραφούν δεδομένα.

Παράδειγμα DMA

- Έστω απαιτείται η μεταφορά 32 Byte από τη δ/νση μνήμης 100 στη συσκευή No4.
- Η CPU θα γράψει **32, 100, 4, Write** στο DMA. Η CPU μπορεί να συνεχίσει να λειτουργεί κανονικά.
- Το DMA αρχίζει να διαβάζει από τη μνήμη και να γράφει στη συσκευή ένα Byte κάθε φορά.
- Στη συνέχεια αυξάνει κατά 1 τη δ/νση μνήμης (νέα δ/νση 101) και μειώνει κατά 1 το υπολειπόμενο μέγεθος μεταφοράς (νέο μέγεθος 31 Byte).
- Μόλις ολοκληρωθεί η μεταφορά ενεργοποιείται μια διακοπή στη CPU.

Διαγράμματα λειτουργίας των τεχνικών Ε/Ε



Διαγράμματα λειτουργίας των τεχνικών Ε/Ε

Τι ονομάζεται υπεξαίρεση κύκλων από τη DMA; Πότε εμφανίζεται;

- Ο ελεγκτής DMA **έχει μεγαλύτερη προτεραιότητα** πρόσβασης στο δίαυλο από ότι η CPU, γιατί η Ε/Ε πρέπει να ολοκληρώνεται όσο το δυνατόν πιο γρήγορα.
- Μια συσκευή υψηλής ταχύτητας (σκληρός δίσκος) με DMA μπορεί να χρησιμοποιεί πολύ εντατικά το δίαυλο για μεταφορά δεδομένων στη μνήμη.
- Η CPU **πρέπει να περιμένει**, αν απαιτείται να διαβάσει κάτι από τη μνήμη. Αυτό καλείται υπεξαίρεση κύκλων.
- Εντούτοις, η μεταφορά DMA έχει πολύ καλύτερες επιδόσεις από κάθε άλλη μεταφορά.

Τι είναι παγίδες;

- Παγίδα (trap) ή εξαίρεση (exception) είναι μια αυτόματη εκτέλεση μιας διαδικασίας η οποία προκαλείται από κάποια συνθήκη που εμφανίζεται στο πρόγραμμά μας (π.χ. διαίρεση με το 0). Σε αυτήν την περίπτωση η εκτέλεση μεταφέρεται σε μια καθορισμένη θέση μνήμης αντί να συνεχιστεί κανονικά.
- Σε αυτή τη θέση μνήμης υπάρχει μια διαδικασία χειριστής παγίδων (trap handler/exception handler), η οποία εκτελεί μια συγκεκριμένη ενέργεια.
- Παραδείγματα παγίδων:
  - Υπερχείλιση, παραβίαση προστασίας, μη ορισμένο opcode, διαίρεση με το μηδέν.

Παραδείγματα από IA32

Vector No.	Mnemonic	Name	Source
0	#DE	Divice Error	IDV and IDIV instrustions
1	#DB	Debug	Any code or data reference
3	#BP	Breakpoint	INT 3 instruction
4	#OF	Overflow	INTO insteuction
5	#BR	BOUND Range Exceeded	BOUND instruction
6	#UD	Invalid Opcode (Undefined Opcode)	UD2 instruction or reserved opcode
7	#NM	Device Not Available(No Math Coprocessor)	Floating-point or WAIT/FWAIT instruction
8	#DF	Double Fault	Any instruction that can generate an exception,an NMI or an INTR
10	#TS	Invalid TSS	Task switsh or TSS access

Τι είναι οι διακοπές;

- Οι διακοπές είναι αλλαγές στη ροή του ελέγχου και σχετίζονται από Ε/Ε.
- Η διακοπή σταματά το εκτελούμενο πρόγραμμα και μεταφέρει τον έλεγχο σε ένα χειριστή διακοπών.
- Ο χειριστής διακοπών επιστρέφει τον έλεγχο στο πρόγραμμα όταν εκτελέσει την ενέργειά του στην ίδια ακριβώς κατάσταση πριν γίνει η διακοπή (δεν έχουν αλλάξει τιμή οι καταχωρητές), (= διαφάνεια διακοπής).
- Οι παγίδες είναι σύγχρονες (με τα ίδια δεδομένα εκτελούνται πάντα στο ίδιο σημείο).
- Οι διακοπές είναι ασύγχρονες (εξαρτάται από την Ε/Ε).

Το διάνυσμα διακοπής

- Σε κάθε συσκευή που επικοινωνεί με διακοπές, αντιστοιχεί ένας ακέραιος αριθμός, που ονομάζεται διάνυσμα διακοπής (interrupt vector).
- Όταν η συσκευή προκαλέσει μια διακοπή, τότε χρησιμοποιείται το συγκεκριμένο διάνυσμα διακοπής της συσκευής για να βρεθεί η διαδικασία εξυπηρέτησης της διακοπής (ISR – Interrupt Service Routine), δηλαδή το κομμάτι του κώδικα που θα εξυπηρετήσει τη διακοπή.
- Απαιτείται **διαφάνεια**, δηλαδή το κομμάτι του κώδικα να μην επηρεάζει την κατάσταση του εκτελούμενου προγράμματος (να μην αλλάζουν καταχωρητές, σημαίες κτλ).

Οι διαφανείς διακοπές

- Διαφανής ονομάζεται η διακοπή της οποίας η εξυπηρέτηση ακολουθεί τους εξής κανόνες:
  - Ο καταχωρητής προγράμματος αποθηκεύεται σε μια γνωστή τοποθεσία, για να επιστρέψει η εκτέλεση μετά τη διακοπή.
  - Όλες οι εντολές μέχρι τη διακοπή έχουν εκτελεστεί.
  - Όλες οι εντολές ΜΕΤΑ τη διακοπή ΔΕΝ έχουν εκτελεστεί. Αν έχουν εκτελεστεί κάποιες (λόγω τεχνικής εκτέλεσης εκτός σειράς) θα πρέπει να αναιρεθούν οι αλλαγές που ίσως έχουν φέρει (π.χ. Σε καταχωρητές).
  - Είναι γνωστή η κατάσταση της εκτέλεσης της εντολής που εκτελείται.

Αντιμετώπιση ταυτόχρονων διακοπών

- Τι θα συμβεί αν ταυτόχρονα υπάρξουν 2 ή παραπάνω διακοπές; Υπάρχουν 2 τεχνικές:
  - Απενεργοποίηση όλων των διακοπών. Απλή λύση, αλλά μπορεί να προκαλέσει πρόβλημα στην Ε/Ε αν υπάρξει μεγάλη καθυστέρηση (μπορεί να γεμίσει η προσωρινή μνήμη π.χ. Modem).
  - Εκχώρηση προτεραιοτήτων. Υψηλή για συσκευές μεγάλης κρισιμότητας και χαμηλή και συσκευές μικρότερης κρισιμότητας. Μόνο υψηλότερης προτεραιότητας μπορούν να προκαλέσουν διακοπή.
  - Απαιτείται **διαφάνεια**.

Κατηγορίες Διακοπών (1/2)

Ως προς την πηγή:

- Διακοπές λογισμικού (προκαλούνται από ειδική εντολή της ISA, π.χ. Int 21h).
- Διακοπές υλικού (προκαλούνται από μια συσκευή Ε/Ε όταν απαιτεί εξυπηρέτηση από τον επεξεργαστή).

Ως προς τη σημαντικότητα:

- Χωρίς μάσκα (non-maskable) δε μπορούν να αγνοηθούν με τίποτα. Απαιτείται άμεση ενασχόληση γιατί εμφανίστηκε καταστροφική περίπτωση (π.χ. σφάλμα ισοτιμίας).
- Με μάσκα (maskable) Μπορούν να απενεργοποιηθούν αν υπάρχει λόγος (π.χ. Χειρισμός άλλης διακοπής).

Κατηγορίες Διακοπών (2/2)

Διάφορες άλλες κατηγοριοποιήσεις:

- **Δια-Επεξεργαστική διακοπή** (inter-processor interrupt): Μια διακοπή που ένας επεξεργαστής στέλνει σε έναν άλλο για να τον ειδοποιήσει για κάποιο θέμα.
- **Ψεύτικη διακοπή** (spurious interrupt): Διακοπή που προκλήθηκε από ανεξήγητη αιτία (π.χ. Ηλεκτρομαγνητική ακτινοβολία, cosmic rays) και όχι από κάποια συσκευή.

- Ως προς τη μορφή:
  - Level-triggered (πυροδότηση στη στάθμη).
  - Edge-triggered (πυροδότηση σε ακμή).

Τα ολοκληρωμένα PIC & APIC

- Η Intel προκειμένου να εξυπηρετεί τις διακοπές και πολλαπλές I/O από τους επεξεργαστές της πρόσθεσε ένα προγραμματιζόμενο ολοκληρωμένο (@1990) με το όνομα **89C59A PIC** (Programmable Interrupt Controller).
- Το 2001 η INTEL δημιούργησε το Advanced PIC (**APIC**) το οποίο υποστήριζε πολλαπλούς επεξεργαστές, 255 διακοπές προγραμματιζόμενες, ταυτόχρονη I/O και μεταβίβαση interrupt σε ελεύθερο επεξεργαστή.
- Απαιτείται υποστήριξη από το λειτουργικό σύστημα (γιατί είναι προγραμματιζόμενο).
- Τα πρώτα χρόνια προκαλούσε πολλά προβλήματα λόγω μη σωστού software/apic drivers.

Σειριακή ή παράλληλη διασύνδεση εξωτερικών συσκευών;

- Η παράλληλη διασύνδεση απαιτεί πολλά καλώδια και μεγαλύτερους συνδετήρες (μεγάλο κόστος).
- Τα καλώδια αυτά ποιο εύκολα καταστρέφονται.
- Απαιτείται ακριβή θωράκιση λόγω EMI και ηλεκτρονικής παρεμβολής.
- Δε μπορούν να επιτευχθούν μεγάλες ταχύτητες γιατί υπάρχει πρόβλημα συγχρονισμού λόγω διαφορετικών ταχυτήτων σε κάθε αγωγό, δυσχεραίνει με το μήκος του καλωδίου.
- Το ολοένα μικρότερο μέγεθος των συσκευών μειώνει το μέγεθος των διεπαφών Ε/Ε.

Η σειριακή μετάδοση έχει υπερισχύσει

- Η σειριακή σύνδεση έχει νικήσει!
- Παραδείγματα:
  - Οι δίσκοι SATA υπερίσχυσαν των PATA.
  - Τα USB καλώδια υπερίσχυσαν έναντι των παράλληλων καλωδίων.
  - Το Firewire υπερίσχυσε έναντι του SCSI.

Ο σωρός στη x86

Τι είναι σωρός;

- Ο σωρός είναι ένα διατεταγμένο σύνολο στοιχείων όπου κάθε φορά μπορούμε να έχουμε προσπέλαση μόνο σε ένα από αυτά. Το σημείο προσπέλασης ονομάζεται κορυφή (top) του σωρού.
- Μόνο στην κορυφή μπορούν να τοποθετηθούν ή να διαγραφούν αντικείμενα.
- Συνήθως υπάρχουν τα εξής στοιχεία:
  - Βάση του σωρού (από που ξεκινάει).
  - Δείκτης του σωρού (ως πιο σημείο έχουμε δεδομένα).
  - Όριο του σωρού (ως πιο σημείο μπορούν να τοποθετηθούν δεδομένα).

Που βρίσκεται ο σωρός;

- Ο σωρός βρίσκεται στην εξωτερική μνήμη.

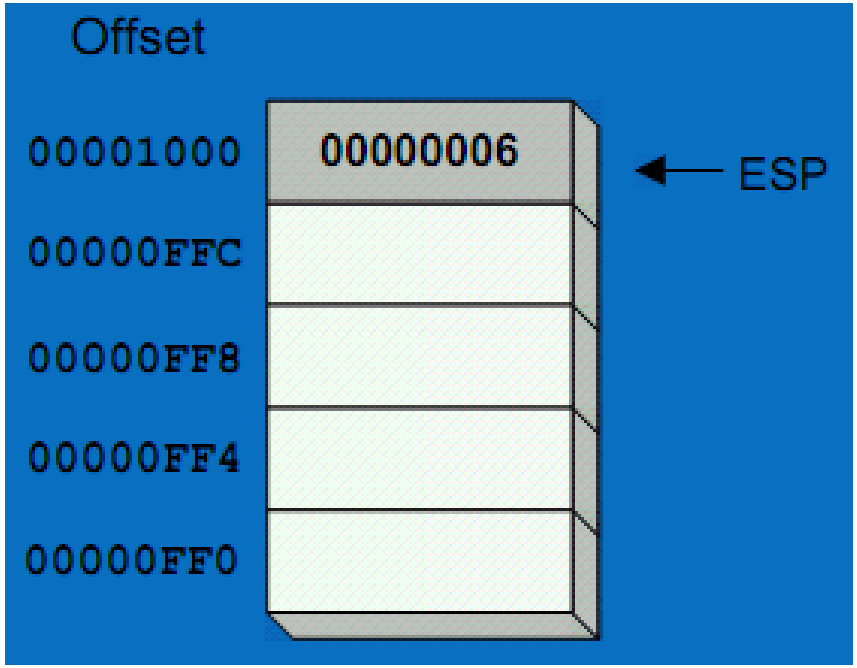
- Η αρχιτεκτονική x86 προσδιορίζει μόνο τη βάση του σωρού (SS), και το δείκτη της κορυφής (SP). Στην αρχιτεκτονική x86, δεν υπάρχει άνω όριο, και για αυτό το λόγο μπορεί να δημιουργηθεί πρόβλημα αν τοποθετηθούν περισσότερα στοιχεία στο σωρό από το μέγεθός του.

Ο Σωρός στη x86

- Είναι μια δομή FILO (First In – Last Out) (σα μια στοίβα από πιάτα).
- Υπάρχουν 2 μόνο λειτουργίες: Εισαγωγή και Εξαγωγή στοιχείων.
- Μπορούμε να εξάγουμε στοιχεία μόνο από το σημείο που δείχνει ο καταχωρητής σωρού (SP stack pointer).
- Όταν τοποθετούμε στοιχεία (εντολή PUSH) ο SP τροποποιείται κατά μέγεθος ίσο με τα στοιχεία που τοποθετούμε (είτε 2 Byte, είτε 4Byte).
- Αν διαβάσουμε στοιχεία από το σωρό (εντολή POP) ο SP τροποποιείται κατά ίσο μέγεθος.

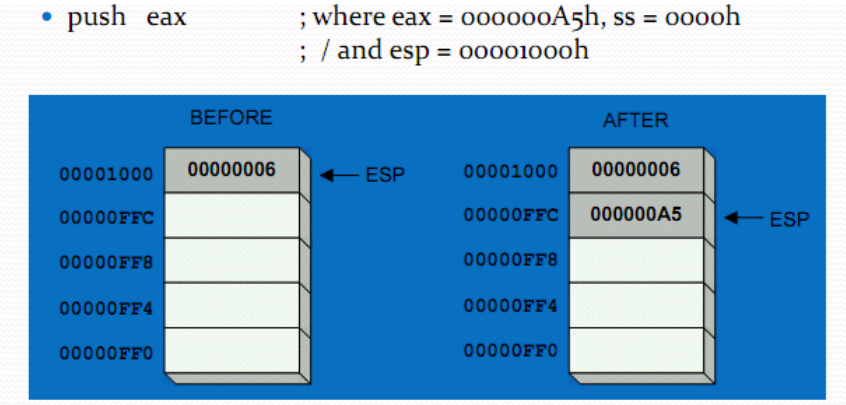
Ο σωρός

- 2 καταχωρητές συνδέονται με το σωρό:
  - SS (stack segment).
  - SP (stack pointer).
- Ο σωρός στην IA32 έχει ομοίως:
  - SS.
  - **E**SP.
- Για να βρούμε την απόλυτη τιμή:SS*10h +(E)SP, η οποία είναι η κορυφή του σωρού.



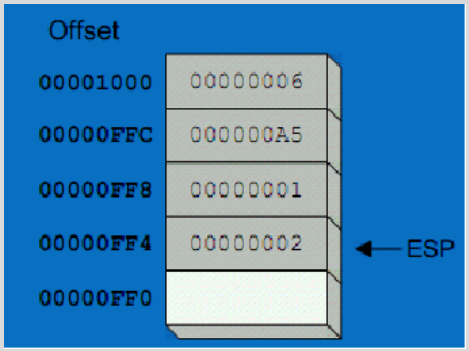
Λειτουργία PUSH (1/2)

- Στη x86 η λειτουργία PUSH γίνεται μόνο με καταχωρητές των 16bit. Μειώνεται ο SP κατά 2 Byte.
- Στην IA32 η λειτουργία PUSH γίνεται μόνο με καταχωρητές των 32bit. Μειώνεται ο SP κατά 4 Byte.



Λειτουργία PUSH (2/2)

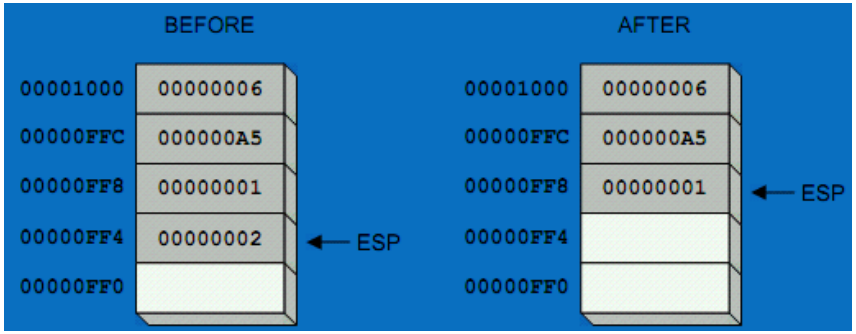
- Ομοίως στην IA32 ο σωρός διαμορφώνεται ως εξής, αν κάνουμε:
- push ebx;where ebx=00000001h & esp=00000FFFCh
- push ecx ; where ecx=00000002h & esp=00000FFF8h



- Ο δείκτης σωρού μειώνεται συνεχώς. Ο σωρός τοποθετείται στο τέλος, για να μην επηρεάζει το πρόγραμμα αν γίνει υπερχείλιση.

Λειτουργία POP

- Αντιγράφεται η τιμή που δείχνει ο SP σε μια μεταβλητή μνήμης ή σε ένα καταχωρητή. Στη συνέχεια αυξάνεται ο SP είτε κατά 4 Byte (για την IA32), είτε κατά 2 Byte (για την x86).
- Παράδειγμαpop edx ; before execution: edx = ? , esp=00000FF4h ; after execution: edx=00000002h, esp=00000FF8.



PUSH & POP

- Στην IA32 υπάρχουν εκτός από τις εντολές PUSH και POP και εντολές που επιτρέπουν να ωθούν και να εξάγουν από/το σωρό:
  - Τις σημαίες (flags) (pushfd, popfd, pushf, popf).
  - Όλους τους καταχωρητές 16bit με μια εντολή (pusha, popa).
  - Όλους τους καταχωρητές 32bit με μια εντολή (pushad, popad).
- Στη x86 δεν υπάρχουν τέτοιες εντολές, αλλά μπορούν να υλοποιηθούν με τις εντολές PUSH, POP.

Η σειρά των PUSH και των POP πρέπει να είναι αντίστροφη

```
push esi                              ; push registers
push ecx
push ebx

mov  esi,OFFSET dwordVal              ; display some memory
mov  ecx,LENGTHOF dwordVal
mov  ebx,TYPE dwordVal
call DumpMem

pop  ebx                              ; restore registers
pop  ecx
pop  esi
```

Χρησιμοποιούμε τα PUSH και POP για εμφωλιασμένο βρόχο

```
    mov ecx,100          ; set outer loop count
L1:                      ; begin the outer loop
    push ecx             ; save outer loop count

    mov ecx,20           ; set inner loop count
L2:                      ; begin the inner loop
    ;
    ;
    loop L2              ; repeat the inner loop

    pop ecx              ; restore outer loop count
    loop L1              ; repeat the outer loop
```

Χρήση του σωρού για να γίνει αντιστροφή ενός string

```
mov ecx, nameSize                          ; Push name on stack
mov esi, 0
L1:  movzx eax, aName[esi]                 ;get character
        push eax                           ;push on stack
        inc esi                            ;point to next character
        loop L1                            ;repeat until entire string
                                           ; /pushed on stack
; Pop the name from stack in reverse, put back in aName

mov ecx, nameSize
mov esi, 0
L2: pop eax                                ;get character
    mov aName[esi], al                     ;store in string
    inc esi                                ;pt. to next character position
    loop L2                                ;repeat until name complete

mov edx, OFFSET aName
call WriteString
call Crlf
```

Σωρός και εμφωλιασμένες διαδικασίες

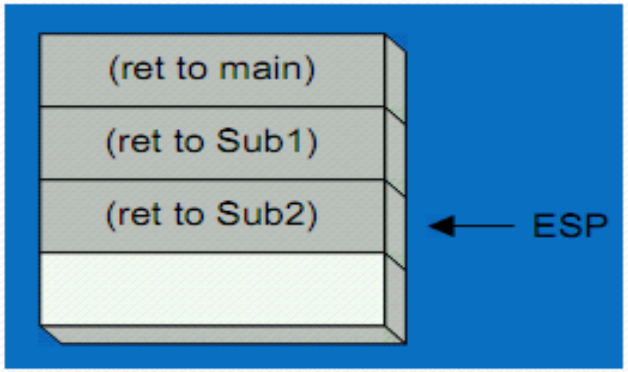
```
main PROC
    .
    .
    call Sub1
    exit
main ENDP

Sub1 PROC
    .
    .
    call Sub2
    ret
Sub1 ENDP

Sub2 PROC
    .
    .
    call Sub3
    ret
Sub2 ENDP

Sub3 PROC
    .
    .
    ret
Sub3 ENDP
```

By the time Sub3 is called, the stack contains all three return addresses:



Source: Kip R. Irvine "Assembly Language for x86 Processors"

Περιορισμοί κλήσεων στη x86 (1/2)

- Τα προηγούμενα παραδείγματα υπέθεταν ότι όλος ο σωρός χρησιμοποιείται από τις κλήσεις διαδικασιών. Αν ο σωρός χρησιμοποιείται και για την προσωρινή αποθήκευση μεταβλητών τότε ο αριθμός των κλήσεων μειώνεται.
- Τι θα γίνει αν υπερβούμε το μέγιστο αριθμό κλήσεων που υποστηρίζει ο σωρός μας;
  - Θα συνεχίζει να μειώνεται ο δείκτης σωρού (SP) και θα δείχνει σε περιοχές μνήμης που ανήκουν σε άλλα τμήματα (π.χ. Τμήμα κώδικα).
  - Αν έχει την ελάχιστη τιμή (0000h) τότε θα γίνει 0FFFFh και θα αρχίσει να διαγράφει τα πρώτα στοιχεία που είχαν τοποθετηθεί σε αυτόν.

Διαδικασίες

Υποστήριξη διαδικασιών ή συναρτήσεων

- Διαδικασία είναι μια ομάδα εντολών η οποία πραγματοποιεί κάποια εργασία και μπορεί να καλείται από πολλά σημεία του προγράμματος.
- **Στην x86 που τοποθετείται ο κώδικας μιας διεργασίας; Πως ορίζεται η αρχή και πως το τέλος κώδικα ρουτίνας;**

Τοποθετείται μέσα στο τμήμα κώδικα (ανάμεσα στις ετικέτες CODE SEGMENT και CODE ENDS). Μπορεί να τοποθετηθεί και σε άλλο τμήμα. Η αρχή ορίζεται με το PROC ενώ το τέλος ENDP όπως παρακάτω:

- MYFUNCTION PROC
- κώδικας
- MYFUNCTION ENDP

Κλήση διαδικασιών

- **Περιγράψτε τη διαδικασία κλήσης συναρτήσεων.**

Μόλις ο επεξεργαστής συναντήσει μια εντολής κλήσης διαδικασίας (CALL), κάτι που συνεπάγεται προσωρινή αλλαγή της επόμενης εντολής εκτέλεσης (τροποποίηση του μετρητή προγράμματος), **πρέπει να αποθηκεύσει την τρέχουσα τιμή του μετρητή προγράμματος** για να γνωρίζει σε ποια εντολή να επιστρέψει όταν ολοκληρωθεί η κλήση της διαδικασίας. Η αποθήκευση της τιμής του μετρητή προγράμματος μπορεί αν γίνει:

- Σε μια θέση μνήμης (πρόβλημα αν κληθεί ξανά).
- Σε έναν καταχωρητή (πρόβλημα αν κληθεί ξανά).
- Στη στοίβα (προτιμώμενο).

Επιστροφή διαδικασιών

- **Περιγράψτε τη διαδικασία επιστροφής ύστερα από κλήση συνάρτησης**

Όταν ο επεξεργαστής εκτελεί μια διαδικασία και συναντήσει την εντολή επιστροφής πίσω στο πρόγραμμα που την είχε καλέσει (εντολή **RET**), τότε τροποποιεί το μετρητή προγράμματος και τοποθετεί τη διεύθυνση επιστροφής που είχε αποθηκεύσει σε κάποιο γνωστό σημείο. Στη συνέχεια ο επεξεργαστής συνεχίζει την εκτέλεση με την επόμενη εντολή που ακολουθεί την κλήση της συνάρτησης.

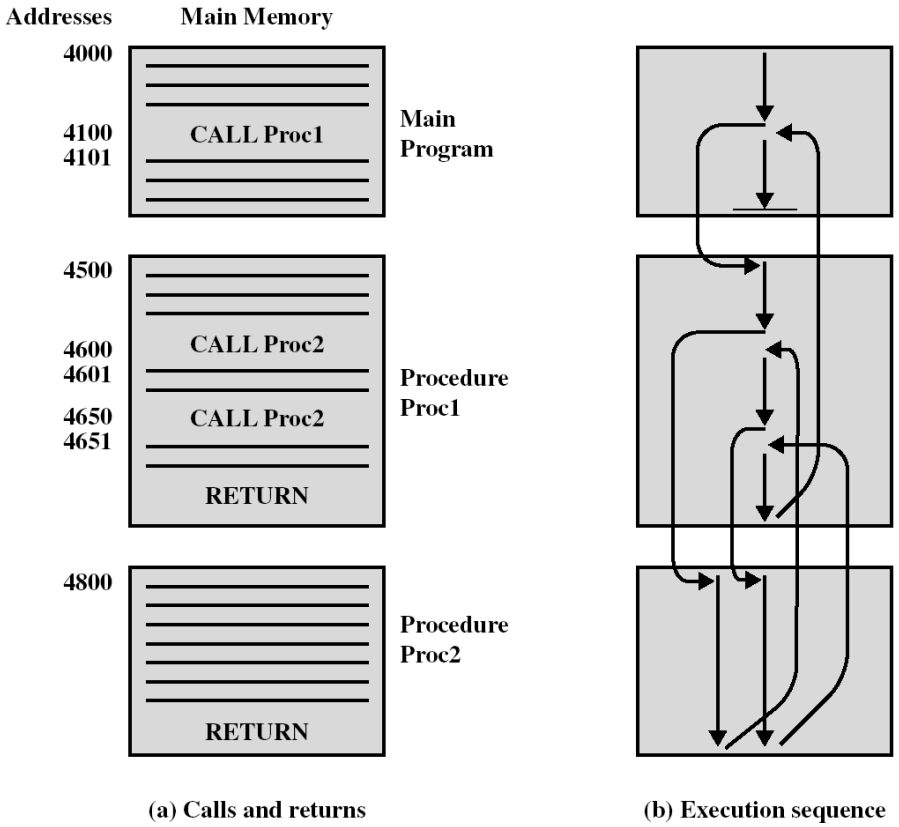
Στοιχεία για τη στοίβα

- Είναι μια περιοχή που βρίσκεται στην εξωτερική μνήμη.
- Ορίζεται από τη βάση (ή κορυφή) και ένα δείκτη που δείχνει στο τελευταίο στοιχείο που έχει εισαχθεί.
- Η στοίβα μπορεί να ξεκινάει από την κορυφή και κάθε φορά που τοποθετείται ένα στοιχείο να μειώνεται ο δείκτης, είτε να ξεκινάει από τη βάση και κάθε φορά που τοποθετείται ένα στοιχείο να αυξάνεται ο δείκτης.

Στοιχεία για τη στοίβα x86

- Η στοίβα στο x86 ξεκινάει από την κορυφή και κάθε φορά που τοποθετείται στοιχείο ο δείκτης μετακινείται προς τη βάση.
- Η στοίβα ξεκινάει από τη διεύθυνση που ορίζει το SS (καταχωρητής τμήματος στοίβας) και ο δείκτης μετατόπισης από την αρχή SP.
- Κάθε εγγραφή στη στοίβα με την εντολή PUSH είναι 2 Byte.
- Κάθε ανάγνωση από τη στοίβα με την εντολή POP είναι 2 Byte.

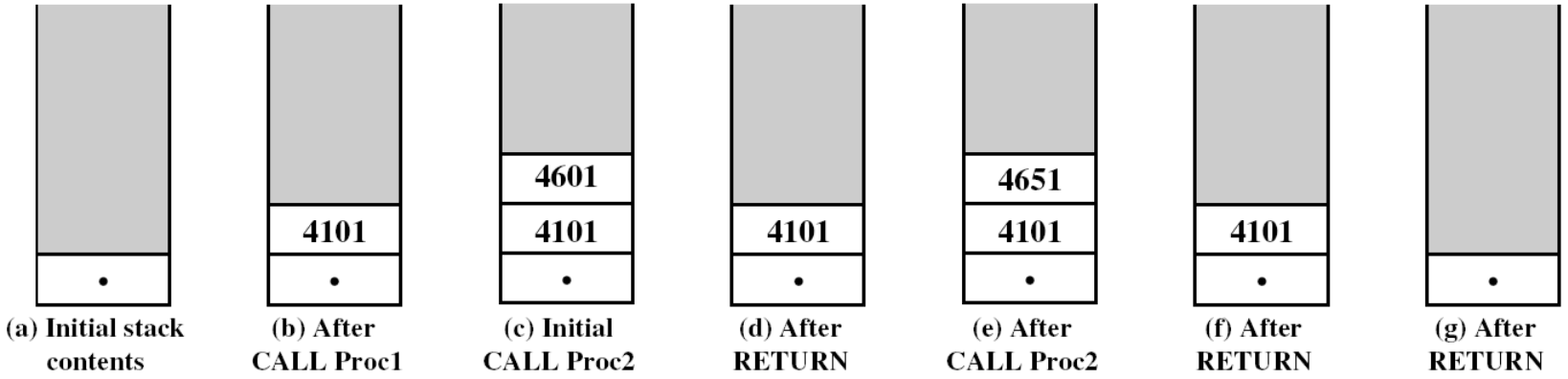
Κλήση διαδικασιών και ροή εκτέλεσης



Παράδειγμα εμφώλευσης διαδικασιών.

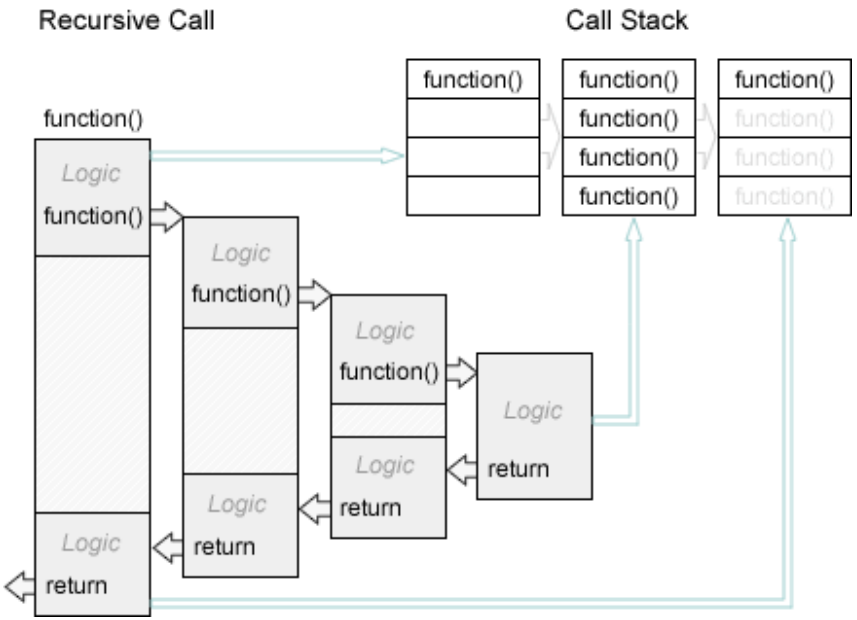
Η χρήση του σωρού σε παράδειγμα

- Κάθε φορά που γίνεται κλήση, τοποθετείται στο σωρό η τιμή του μετρητή προγράμματος.
- Κάθε φορά που επιστρέφουμε από κλήση, ο μετρητής προγράμματος παίρνει την τιμή που βρίσκεται στο σωρό.



Κλήση και επιστροφή διαδικασιών

- Σε κάθε κλήση διαδικασίας αποθηκεύεται η διεύθυνση επιστροφής.
- Όταν ολοκληρωθεί η διαδικασία τότε αφού διαβαστεί η δ/νση επιστροφής και τοποθετηθεί στο μετρητή προγράμματος, απομακρύνεται από το σωρό.



Υλοποίηση διαδικασιών στη x86 (1/2)

Απαιτούνται:

(α) Να οριστεί ένα τμήμα μνήμης για το σωρό για την αποθήκευση της δ/νσης επιστροφής της συνάρτησης:

```
SOROS SEGMENT STACK
db 256 dup(0)
SOROS ENDS
```

(β) Ο κώδικας της συνάρτησης ξεκινάει με **(όνομα συνάρτησης) PROC** και τελειώνει με **RET** (επιστροφή) και **(όνομα συνάρτησης):**

```
ENDP
SYNARTHSHMOU PROC
κώδικας
RET
SYNARTHSHMOU ENDP
```

Υλοποίηση διαδικασιών στη x86 (2/2)

- Στο σημείο που θέλουμε να καλέσουμε τη συνάρτηση, την καλούμε με την εντολή CALL (όνομα συνάρτησης).

```
Κώδικας
CALL SYNARTHSHMOU
κώδικας
```

- Αν η συνάρτηση τροποποιεί καταχωρητές επιβάλλεται να αποθηκεύσουμε τις τρέχουσες τιμές που έχουν στην είσοδο της συνάρτησης και να τις επαναφέρουμε στην έξοδο. Μπορούμε να τις αποθηκεύσουμε σε θέσεις μνήμης ή στο σωρό.

Αναδρομική συνάρτηση

- Αναδρομική συνάρτηση είναι η συνάρτηση η οποία μπορεί να καλέσει τον εαυτό της και να συνεχίζει να λειτουργεί ορθά.
- Προσοχή στο που αποθηκεύεται η δ/νση επιστροφής, ώστε να μη διαγράφεται σε κάθε καινούργια κλήση. Για αυτό το λόγο προτιμάται ο σωρός ως θέση αποθήκευσης της μνήμης.
- (**έμμεση** ή **αλυσιδωτή**) Αναδρομική διαδικασία είναι και όταν η Α συνάρτηση θα καλέσει τη Β συνάρτηση, που θα καλέσει την C συνάρτηση που θα καλέσει την Α συνάρτηση.

Αποθήκευση της δ/νσης επιστροφής στο x86

- Στη x86 αρχιτεκτονική μόλις γίνει κλήση συνάρτησης, τοποθετείται η δ/νση επιστροφής στο σωρό (stack).
  - Αν η κλήση είναι μέσα στο ίδιο τμήμα κώδικα με τη call τότε τοποθετείται στο σωρό το IP (2Byte) και θεωρείται κοντινή (near) ή ενδοτμηματική κλήση.
  - Αν η κλήση είναι σε άλλο τμήμα κώδικα (όπως π.χ. Με τη κλήση int 21h) τότε τοποθετείται στο σωρό το CS και το IP (συνολικά 4 Byte) και θεωρείται μακρινή (far) κλήση.
  - Αναλόγως του είδους της κλήσης, υπάρχει και η κοντινή επιστροφή (near) RET ή η μακρινή επιστροφή (far) RET. Ο assembler καθορίζει αν θα είναι far / near.
  - Επίσης, υπάρχει και η επιστροφή από INT (iret).

Περιορισμοί κλήσεων στη x86 (2/2)

- Αν ορίσουμε το σωρό ως **db 256 dup(0)**, σημαίνει ότι ορίζουμε 256 Byte σωρού. Πόσες κλήσεις συνεχόμενες (χωρίς να επιστρέψουμε) υποστηρίζονται χωρίς πρόβλημα αν είναι far ή near;
  - Κατά τη μακρινή κλήση αποθηκεύονται 4 byte στο σωρό, οπότε συνολικά 64 κλήσεις (64*4=256).
  - Κατά την κοντινή κλήση αποθηκεύονται 2 Byte στο σωρό, οπότε συνολικά 128 κλήσεις.
- Πόσες κλήσεις χωρίς επιστροφή υποστηρίζονται το μέγιστο στη x86;
  - Στη x86 το μέγιστο μέγεθος τμήματος είναι 65535 Byte.

Τέλος Ενότητας