X86 Assembly – Reverse Eng

Λίγα και καλά πραγματάκια

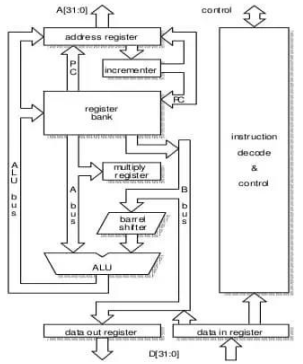
Τι είναι η Assembly;

- Γενικά, όπως ξέρετε ο υπολογιστής διαβάζει ρεύμα ως bits (0, 1). Αυτά είναι και τα μόνα ψηφία που καταλαβαίνει
- Επειδή ο προγραμματισμός με άσσους και μηδενικά είναι αδύνατος, οι λόου λέβελ τυπάδες σκέφτηκαν: Κάτσε ρε φίλε, αφού έχω φτιάξει κάποιο ρημαδοχάρντγουερ με συγκεκριμένες λειτουργίες, να μη φτιάξω μια απλή και περιεκτική γλώσσα που να τις περιγράφει?
- Έτσι δημιουργήθηκε η Assembly

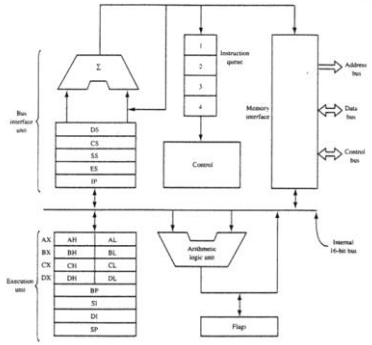
Άρα υπάρχει μία Assembly

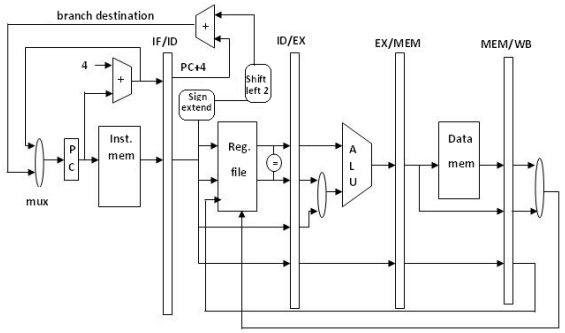
- Εμμμ, όχι?
- Όπως είπαμε, η assembly περιγράφει τις λειτουργίες του συστήματος. Ε, όπως φαντάζεστε, ανάλογα με την **αρχιτεκτονική** έχουμε και άλλη assembly ☹
- Βέβαια, η assembly δεν είναι ανάγκη να αφορά μόνο αρχιτεκτονική (βλέπε python bytecode, .NET Assembly)

The ARM Architecture



X86 Architecture





MIPS Architecture

Σημαντικά στοιχεία μιας αρχιτεκτονικής

- **Καταχωρητές** → Δείτε τους σαν μίνι αποθηκευτικούς χώρους. Αποθηκεύουν αριθμητικές τιμές και διευθύνσεις μνήμης προσωρινά και κάνουν πράξεις με αυτές
- **Μνήμη** → Επειδή ένα πρόγραμμα έχει εντολές και δεδομένα και όλα αυτά δε χωράνε στα λίγα bytes των καταχωρητών, υπάρχει μια ή περισσότερες μονάδες μνήμης που είναι υπεύθυνες για την διαχείριση των παραπάνω.
- Οκέι επίσης υπάρχουν αριθμητικές και λογικές μονάδες (ALU) που υλοποιούν τη λογική των διαφόρων πράξεων, καθώς και μονάδες εισόδου, εξόδου δεδομένων (πχ για περιφερειακές συσκευές), μονάδες ελέγχου κτλ., αλλά θα ‘θελα για τώρα να μείνουμε λίγο στη μνήμη και στους καταχωρητές

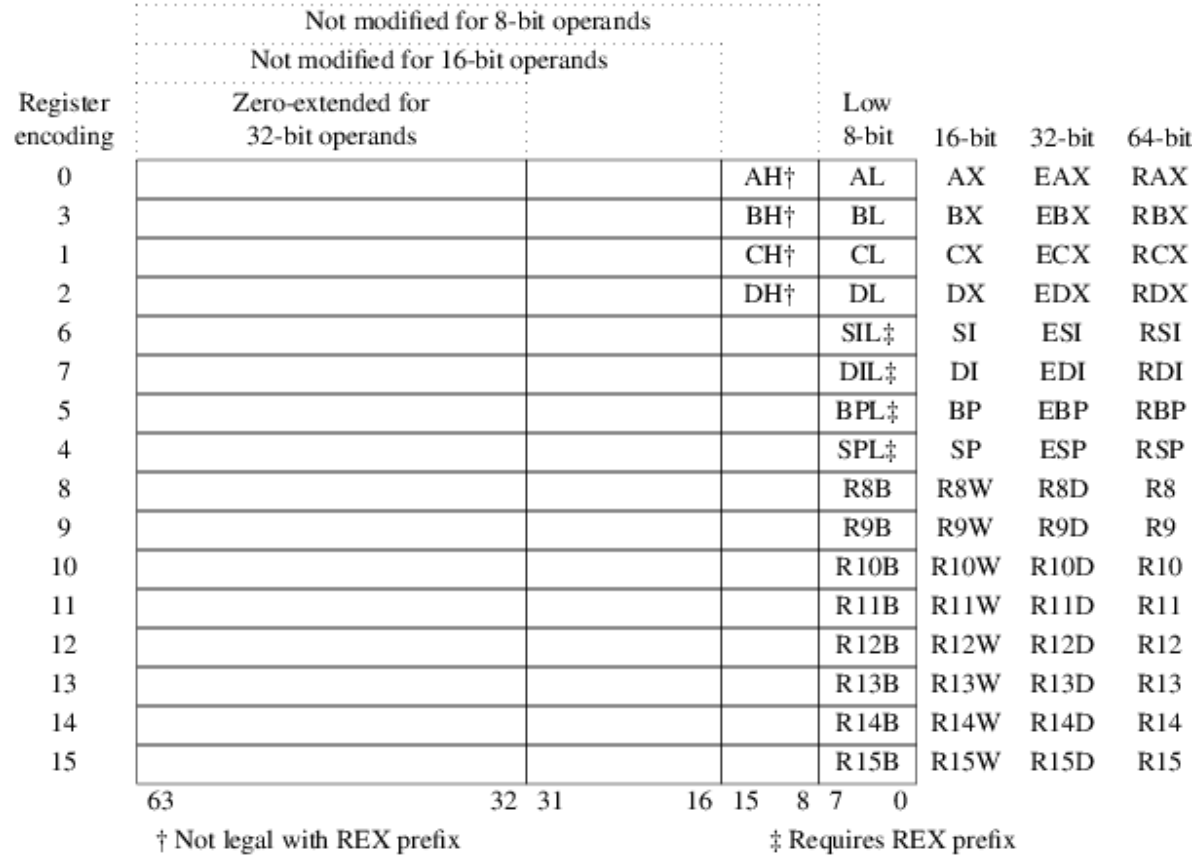
x86 Assembly

- Developed by Intel
- Λέγεται x86 καθώς βασίστηκε στον επεξεργαστή Intel 8086
- Ναι, όσοι έχετε Windows στο PC σας, λογικά έχετε x86 αρχιτεκτονική συνόλου εντολών (instruction set architecture - ISA)*
- Πλέον συναντάμε 2 είδη x86 ISA την σκέτη x86 που αναφέρεται σε 32-bit συστήματα και την x86_64 ή σκέτο x64 που αναφέρεται σε 64-bit συστήματα
- N-bit αρχιτεκτονική: Μπορεί να επεξεργαστεί δεδομένα το πολύ N-bits. Για παράδειγμα, σε μια 32-bit αρχιτεκτονική, ο αριθμός 0x1234567890 δεν μπορεί να φορτωθεί σε κάποιον καταχωρητή και άρα να χρησιμοποιηθεί για κάποια πράξη, καθώς έχει μήκος 5 bytes (40 bits). Αντίθετα σε μια 64-bit αρχιτεκτονική αυτό είναι εφικτό.

*The instruction set architecture (ISA) dictates what instructions are physically possible on a given CPU. The assembly language may include some pseudo-instructions, that the assembler converts into appropriate real (ISA) instructions

Καταχωρητές (registers)

- Η εικόνα των καταχωρητών στην x86 Assembly είναι αυτή που φαίνεται στο σχήμα δεξιά.
- Επαναλαμβάνω ότι οι καταχωρητές είναι θεμελιώδης μονάδα ενός προγράμματος και μιας αρχιτεκτονικής
- Από το σχήμα στα δεξιά λείπει ο καταχωρητής rip που πρόκειται για τον program counter (ουσιαστικά δείχνει σε εντολές του προγράμματος)



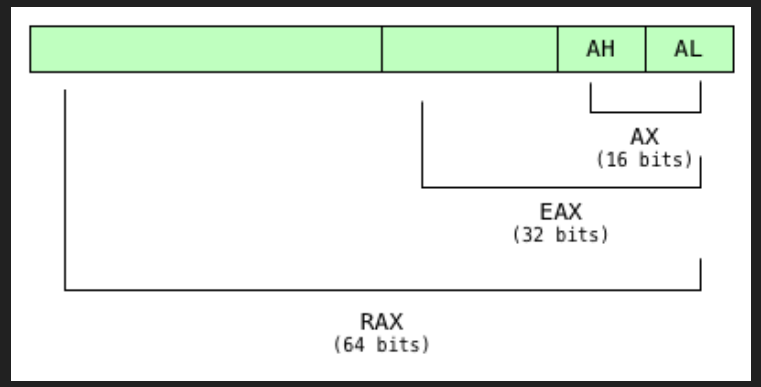
Register encoding	Zero-extended for 32-bit operands	Not modified for 16-bit operands	Not modified for 8-bit operands		Low 8-bit	16-bit	32-bit	64-bit
0				AH†	AL	AX	EAX	RAX
3				BH†	BL	BX	EBX	RBX
1				CH†	CL	CX	ECX	RCX
2				DH†	DL	DX	EDX	RDX
6					SIL‡	SI	ESI	RSI
7					DIL‡	DI	EDI	RDI
5					BPL‡	BP	EBP	RBP
4					SPL‡	SP	ESP	RSP
8					R8B	R8W	R8D	R8
9					R9B	R9W	R9D	R9
10					R10B	R10W	R10D	R10
11					R11B	R11W	R11D	R11
12					R12B	R12W	R12D	R12
13					R13B	R13W	R13D	R13
14					R14B	R14W	R14D	R14
15					R15B	R15W	R15D	R15
	63	32	31	16	15 8	7 0		

† Not legal with REX prefix

‡ Requires REX prefix

Λίγα παραπάνω λογάκια για τους καταχωρητές

- Ορίστε μια εικόνα του καταχωρητή rax visualized:



- Ανάλογα με το πόσα bits του καταχωρητή θα χρησιμοποίησουμε μιλάμε για
  - ax → 2 bytes (16 bits)
  - eax → 4 bytes (32 bits)
  - rax → 8 bytes (64 bits, προφανώς rax υπάρχει μόνο σε 64-bit αρχιτεκτονικές)
  - Συγκεκριμένα για τον ax μπορούμε επίσης να προσπελάσουμε το MSB, ah, και το LSB, al

Εντολές Assembly

- Οι εντολές assembly στην πιο απλή τους μορφή χαρακτηρίζονται από μια χαρακτηριστική λέξη (mov, add, sub, div, mul), μία πηγή και έναν προορισμό.
- Γενικά χρησιμοποιούμε το intel syntax (Εντολή + προορισμός + πηγή)
- Για παράδειγμα, οι εντολές:

mov rax, rbx, μεταφράζεται ως μετακίνησε (mov) στον καταχωρητή rax (destination) την τιμή του rbx (source)

add rax, 5, μεταφράζεται ως πρόσθεσε (add) στον καταχωρητή rax (destination) την τιμή 5 (source)

- Προφανώς κάθε εντολή έχει τη δική της σύνταξη, οπότε καλό είναι να κοιτάμε το documentation

Quiz time

- Σε κάθε ένα από τα παρακάτω παραδείγματα, ο καταχωρητής eax έχει αρχικά την τιμή 0x40000000

- mov eax, 0x13371337 // eax = ?
- mov eax, 0x4a // eax = ?
- mov ax, 0x1337 // eax = ?
- mov ah, 0x13 // eax = ?
- mov ebx, 0x13371337
  mov eax, ebx // eax = ?
- add eax, 0x1337 // eax = ?
- sub eax, 0x40000001 // eax = ?

- mov ebx, 2
  mul eax // eax = ?
- mov ebx, 2
  div eax // eax = ?
- start: // label, δεν εκτελείται
  sub eax, 0x1
  jne start
  finish:
  ret // eax = ?

Καλό είναι να κρατήσετε το εξής

- Όταν λέμε **mov eax, 0x05** στην πραγματικότητα εννοούμε **mov eax, 0x00000005**.
- Ανάλογα με το μέγεθος του καταχωρητή που θα χρησιμοποιήσουμε, αν η τιμή που φορτώνουμε σε αυτόν αναπαρίσταται με λιγότερα bytes από ότι το μέγεθός του (π.χ. εδώ το 0x05 είναι 1 byte, ενώ ο eax έχει μέγεθος 4 bytes), τότε τα υπόλοιπα bytes του καταχωρητή γεμίζουν με μηδενικά. Έτσι, αν ο eax είχε αρχικά την τιμή eax = 0x40000000, στο τέλος θα έχει την τιμή eax = 0x00000005.
- Αντίστοιχα η εντολή **mov ax, 0x05** μεταφράζεται σε **mov ax, 0x0005**. Έτσι, όπως είδατε αν ο eax έχει αρχικά την τιμή eax = 0x4000**0000**, τότε, με τον ax προσπελάζουμε **μόνο** τα δύο τελευταία bytes και η τελική τιμή του eax = 0x40000005

Μνήμη

- Εκτός που δίνουμε με άμεσο τρόπο στο πρόγραμμα (π.χ. όπως στην προηγούμενη διαφάνεια), μπορούμε να αποθηκεύσουμε και να φορτώσουμε δεδομένα από τη μνήμη
- Για να προσπελάσουμε τη μνήμη μέσω assembly χρειαζόμαστε 2 πράγματα
  1. Να φορτώσουμε τη διεύθυνση της θέσης μνήμης που θέλουμε να προσπελάσουμε σε κάποιον καταχωρητή.
  2. Να χρησιμοποιήσουμε το **«[ ]»** notation που υποδηλώνει προσπέλαση μνήμης
- Κάθε θέση μνήμης χωράει **1 byte** (ωστόσο μπορούμε να προσπελάσουμε κάποιες συνεχόμενες θέσεις μνήμης ταυτόχρονα όπως θα φανεί στα παραδείγματα)
- Η μνήμη ακολουθεί little endian order (βλέπε διαφάνειες NTUA_HACK)

Παράδειγμα

Έστω ότι θέλουμε να αποθηκεύσουμε την τιμή 0x87 στη θέση μνήμης 0x40500010
Ένας τρόπος να το κάνουμε αυτό είναι ο ακόλουθος:

```
mov ebx, 0x40500010
mov al, 0x87
mov [ebx], al
```

ή

```
mov ebx,0x40500010
mov byte ptr [ebx], 0x87
```

Προφανώς γίνεται και το αντίστροφο (φορτώνω από τη θέση μνήμης 0x40500010 στον al):

```
mov ebx, 0x40500010
mov al, [ebx]
```

Quiz time

- Σε κάθε μία από τις παρακάτω ερωτήσεις ζητάμε την τιμή της μνήμης στη θέση 0x40500004. Θεωρήστε ότι ο καταχωρητής ebx έχει αρχικά την τιμή 0x40500004 και ότι η μνήμη στη θέση 0x40500004 έχει αρχικά την τιμή 0x69

- mov al, 0x20
  mov [ebx], al // memory = ?

- mov eax, 0x12345678
  mov [ebx], eax // memory = ?
  // Hint: little endian

- mov eax, -2
  mov ecx, 0x12345678
  mov [ebx + eax], ecx // memory = ?

- mov byte ptr [ebx], 0x40 // memory = ?

- mov ebx, 0x40500003
  mov dword ptr [ebx], 0x40 // memory = ?

- mov ebx, 0x40500000
  mov qword ptr [ebx], 0x12345678 // memory = ?

Ας δούμε πώς ένα αλγοριθμικό παράδειγμα θα μπορούσε να μεταφραστεί σε x86 assembly

- **Problem**: Έχεις πάει για ελιές οκέι; Μπροστά σου έχεις άπειρα δέντρα με ελιές, 2 καλάθια Α,Β χωρητικότητας 64 ελιών το καθένα και ένα αγροτικό Μ, που μπορείς να χύνεις άφοβα ελιές όταν γεμίζουν τα καλάθια. Ο στόχος είναι να μαζέψεις Ν ή περισσότερες ελιές συνολικά.

  Όπως καταλαβαίνετε δεν έχω πάει για ελιές στη ζωή μου :(.

- Ένας αλγόριθμος για να μαζέψεις τις ελιές είναι ο εξής:

```
num_iter = (N /64) * 2
Επανάλαβε num_iter φορές:
    Γέμισε τα καλάθια Α,Β
    Άδειασε τα καλάθια Α,Β στο Μ
```

Πώς μεταφράζεται αυτό σε assembly?

- Έστω ότι 1 ελιά = 1 byte και συγκερκιμένα το byte 0xaa αντιστοιχεί σε μια ελιά.
- Έστω ότι ο συνολικός αριθμός των ελιών που πρέπει να μαζέψουμε είναι αποθηκευμένος στον καταχωρητή eax.
- Τα καλάθια θα τα αντιπροσωπεύσουμε με τους καταχωρητές ebx και ecx
- Το αγροτικό θα αντιστοιχηθεί στη μνήμη δεδομένων του προγράμματος στην οποία θα αποκτάμε πρόσβαση μέσω του καταχωρητή edi. Έστω ότι η αρχική διεύθυνση του edi είναι η 0x401000.
- Τότε το παραπάνω πρόγραμμα μεταφράζεται στο να γεμίσουμε N bytes της μνήμης προγράμματος με το byte 0xaa
- Ακολουθέι η assembly με σχόλια:

Το ίδιο σε assembly

```
// Αρχικά eax ← N, edi ← 0x401000

mov r8, 32
div  r8                  // eax = eax / 32 (χωρητικότητα)
mov r8,2
mul r8                   // eax = eax*2 (πλήθος καταχωρητών)

// Ο eax πλέον αντιστοιχεί στον μετρητή των επαναλήψεων

collect_olives:     // ετικέτα

        mov ebx, 0xaaaaaaaa        // φόρτωσε 4 ελιές στον ebx
        mov ecx, 0xaaaaaaaa        // φόρτωσε 4 ελιές στον ecx

        mov [edi], ebx             // βάλε 4 ελιές στη μνήμη
        add edi,  4                // αύξησε τον δείκτη στη μνήμη κατά 4 θέσεις

        mov [edi], ecx             // βάλε 4 ελιές στη μνήμη
        add edi,  4                // αύξησε τον δείκτη στη μνήμη
                                   //κατά 4 θέσεις

        sub eax, 1                 // Μείωσε τον μετρητή
                                   // επαναλήψεων κατά 1
        jne collect_olives               // Αν δεν έχει μηδενιστεί
                                         // επανάλαβε το loop

        ret
```