ΗΥ 232
Οργάνωση και Σχεδίαση Υπολογιστών

Intel x86 ISA

Νίκος Μπέλλας
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών ΗΥ

RISC vs. CISC

- H assembly των επεξεργαστών ARM, SPARC (Sun), και Power (IBM) είναι όμοιες με την assembly του MIPS.
  - Αρχιτεκτονική RISC (Reduced Instruction Set Architectures)

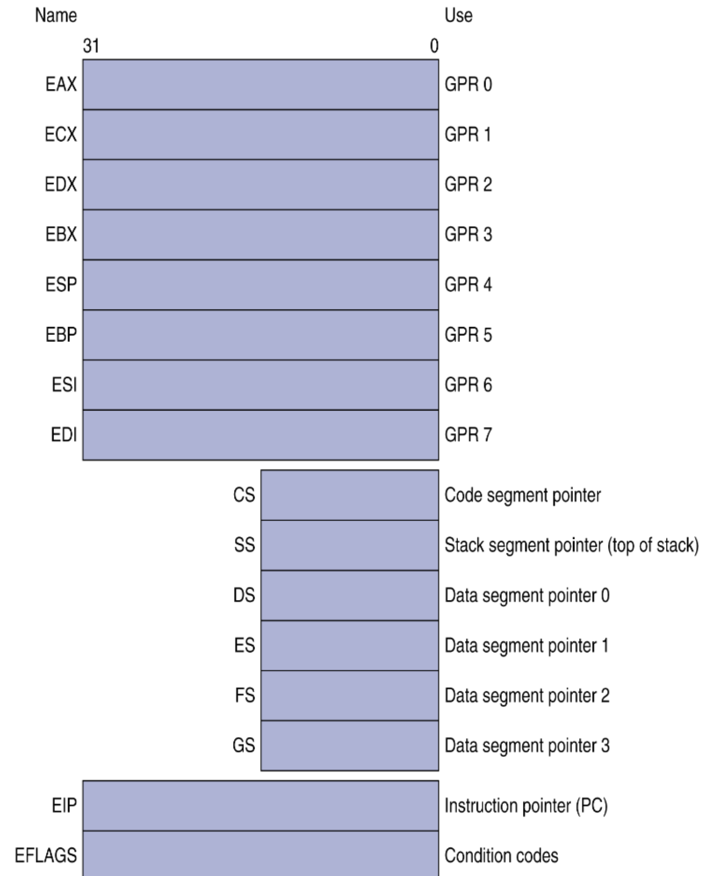
- H assembly της σειράς των επεξεργαστών Intel 80x86 είναι κάπως διαφορετική.
  - Αρχιτεκτονική CISC (Complex Instruction Set Architectures)

Σύγκριση x86 και MIPS

- Οι δύο αυτές αρχιτεκτονικές έχουν πολλά κοινά χαρακτηριστικά.
  - Έχουν καταχωρητές και byte-addressable μνήμες.
  - Οι πιο πολλές εντολές είναι όμοιες (αριθμητικές, λογικές, μεταφοράς, διακλάδωσης)
- Αλλά και διαφορές (διαφορές του x86, 32 bits ή IA-32)
  - Λιγότεροι καταχωρητές (8) με διαφορετικά ονόματα και λιγότερο γενικού σκοπού. O x86-64 έχει 16 καταχωρητές.
  - Η έλλειψη πολλών καταχωρητών έχει ως αποτέλεσμα πολύ μεγαλύτερη χρήση της στοίβας που είναι τμήμα της αρχιτεκτονικής.
  - O x86 χρησιμοποιεί 2 I/O operands, ενώ ο MIPS 3
  - Πράξεις με έμμεση χρήση της μνήμης: Memory + Reg → Reg. Κύρια ίσως διαφορά μεταξύ CISC και RISC
  - Εντολές διακλάδωσης με χρήση conditional flags
  - Εντολές με διαφορετικό μήκος (από 1 μέχρι και 17 bytes)

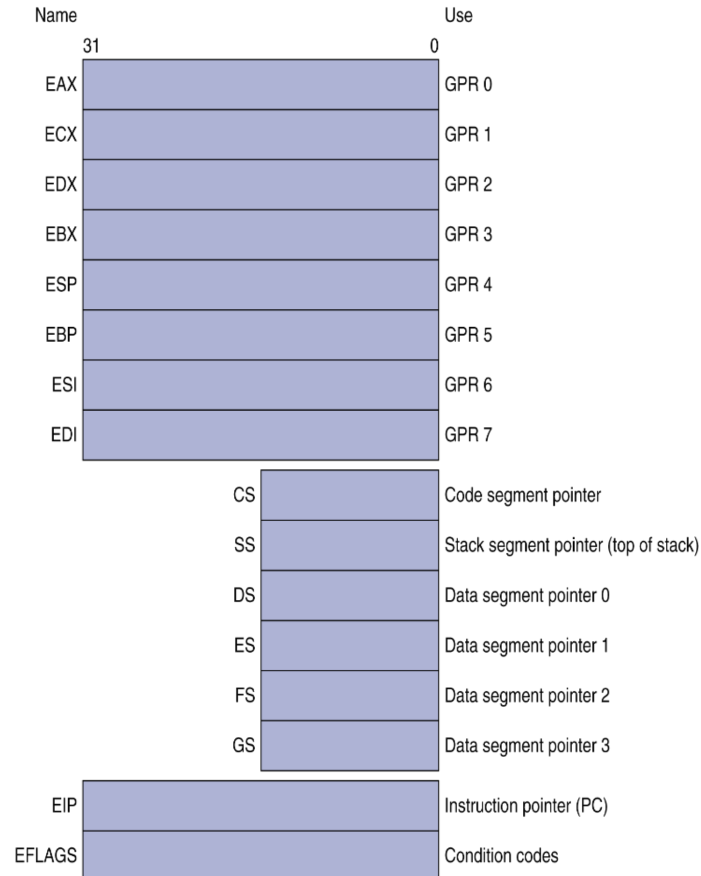
Καταχωρητές του IA-32

- Λιγότεροι και ειδικού σκοπού
  - 8 καταχωρητές ακεραίων
    - **eax, ebx, ecx, edx** (γενικού σκοπού)
    - **ebp** (frame pointer)
    - **esp** (stack pointer)
    - **esi** (extended source index)
    - **edi** (extended destination index)
  - Ειδικού σκοπού
    - **eip** (Instruction Pointer – PC)
    - **eflags** (Conditional Flags)
  - Αντίθετα με τον MIPS, δεν μπορούν όλες οι εντολές να χρησιμοποιήσουν όλους τους καταχωρητές



Καταχωρητές του IA-32

- Δεν υπάρχει πολύς χώρος για temporary values σε πράξεις
  - Ο x86 χρησιμοποιεί κώδικα 2 τελεστέων (operands)
  - **op x, y** # y = y op x
- Πολύ δύσκολο για τον compiler (ή τον προγραμματιστή) να αποθηκεύσει όλες τις μεταβλητές σε καταχωρητές.
- Χρήση της στοίβας για αυτόν τον λόγο.
- *Ο x86-64 έχει 8+8 καταχωρητές των 64 bits*
  - *rax, .., rdi, r8, r15*



Η στοίβα στον x86

- Η στοίβα είναι κομμάτι της μνήμης με ειδικό status
- Ο καταχωρητής **esp** είναι ο stack pointer
- Εντολές **push** και **pop** μόνο για την στοίβα:
  - **push %eax** # ESP = ESP – 4. M[ESP] = EAX
  - **pop %ecx** # ECX = M[ESP]. ESP = ESP + 4
  - *Όπως ακριβώς και στον MIPS, η στοίβα μεγαλώνει από μεγαλύτερες προς μικρότερες διευθύνσεις.*
- Η εντολή **call** (η αντίστοιχη του **jal**) σπρώχνει την διεύθυνση επιστροφής στο stack
  - **call label** # ESP = ESP – 4. M[ESP] = EIP+5. EIP = label
- Η στοίβα χρησιμοποιείται και για να περνούμε παραμέτρους σε συναρτήσεις
  - Δεν υπάρχουν καταχωρητές $a0, $a1, κοκ όπως στον MIPS.
- Ο καταχωρητής **ebp** είναι ο frame pointer (ή base pointer) και δείχνει στην αρχή του frame μιας συνάρτησης
- Χρησιμοποιείται ευρέως γιατί δεν αλλάζει συχνά τιμή όπως ο **esp**

Ένα απλό παράδειγμα

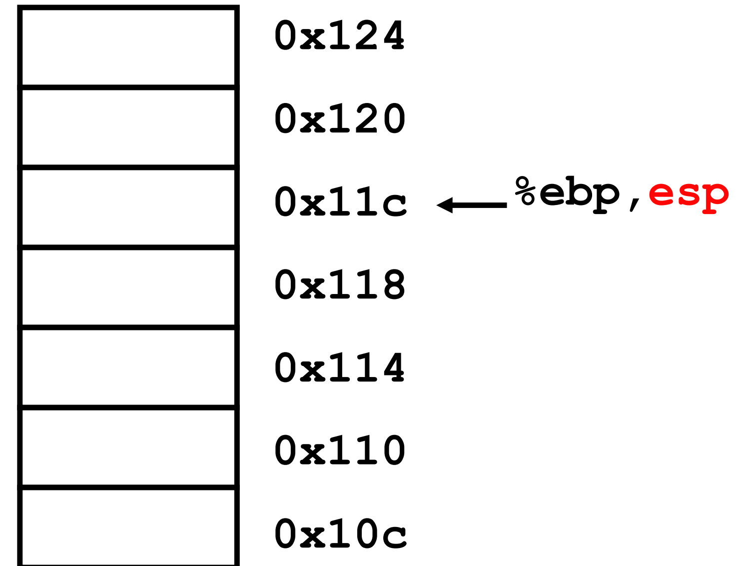
```
int main() {
  int one = 123, two = 456;
  swap(&one, &two);
  ...
}
```

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

Ένα απλό παράδειγμα

```
int main() {
  int one = 123, two = 456;
  swap(&one, &two);
  ...
}
```

```
…
subl     $8, %esp
movl    $123, -8(%ebp)
movl     $456, -4(%ebp)
leal     -4(%ebp), %eax
pushl   %eax
leal     -8(%ebp), %eax
pushl   %eax
call     swap
...
```

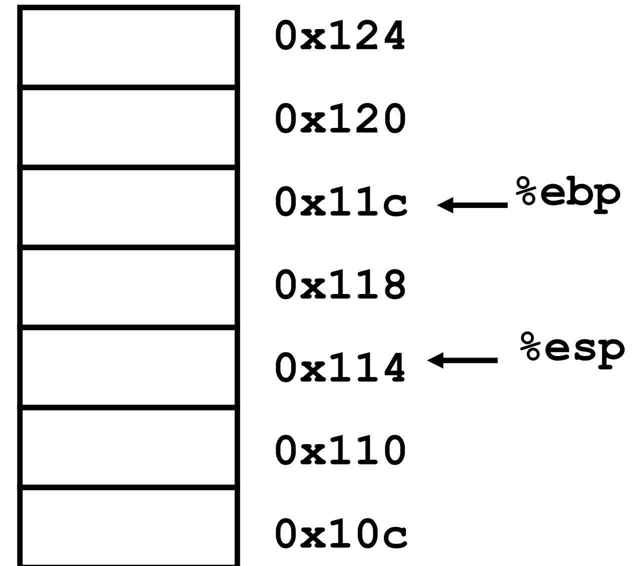


Key:
sub = subtract
l = long (32-bit operation)
$8 = literal 8
%esp= stack pointer register
ESP = ESP - 8

Ένα απλό παράδειγμα

```
int main() {
   int one = 123, two = 456;
   swap(&one, &two);
   ...
}
```

```
…
subl      $8, %esp
movl      $123, -8(%ebp)
movl      $456, -4(%ebp)
leal      -4(%ebp), %eax
pushl     %eax
leal      -8(%ebp), %eax
pushl     %eax
call      swap
...
```

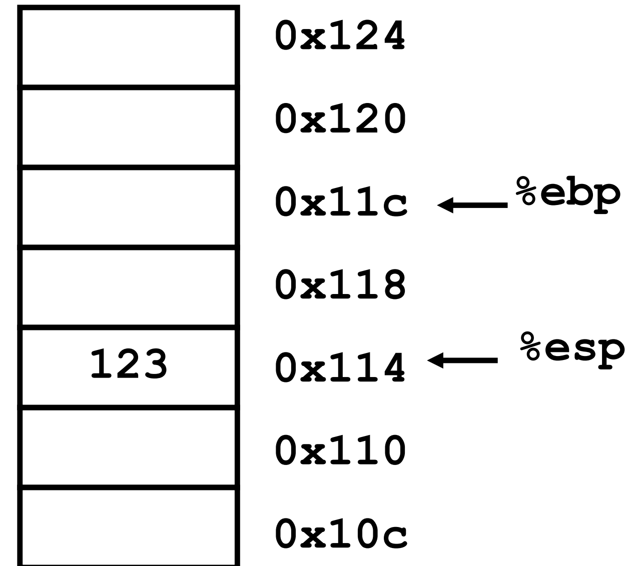


Key:
mov = data movement
l = long (32-bit operation)
$123 = literal 123
-8(%ebp) = base + offset addressing
M[EBP - 8] = 123

Ένα απλό παράδειγμα

```
int main() {
  int one = 123, two = 456;
  swap(&one, &two);
  ...
}
```

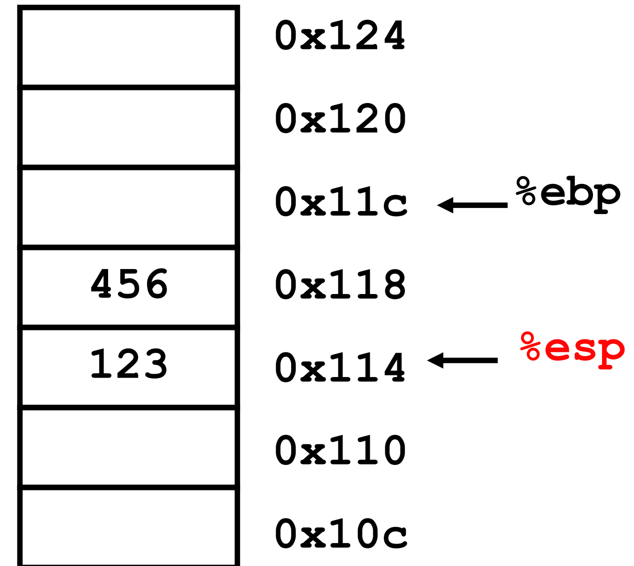
```
…
subl     $8, %esp
movl     $123, -8(%ebp)
movl     $456, -4(%ebp)
leal     -4(%ebp), %eax
pushl    %eax
leal     -8(%ebp), %eax
pushl    %eax
call     swap
...
```



Ένα απλό παράδειγμα

```
int main() {
  int one = 123, two = 456;
  swap(&one, &two);
  ...
}
```

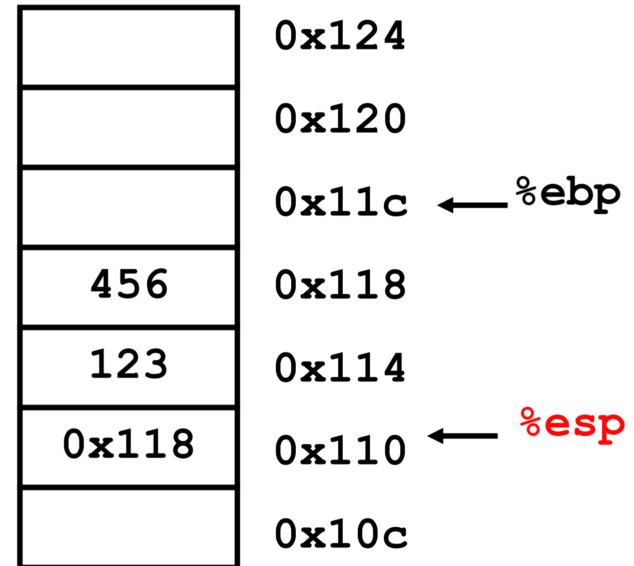
```
…
subl      $8, %esp
movl     $123, -8(%ebp)
movl      $456, -4(%ebp)
leal      -4(%ebp), %eax
pushl    %eax
leal      -8(%ebp), %eax
pushl    %eax
call      swap
...
```



Key:
(push arguments in reverse order)
lea = load effective address
(don't do a load, just compute addr.)
EAX = EBP - 4
M[ESP - 4] = EAX
ESP = ESP - 4

Ένα απλό παράδειγμα

```
int main() {
  int one = 123, two = 456;
  swap(&one, &two);
  ...
}
```

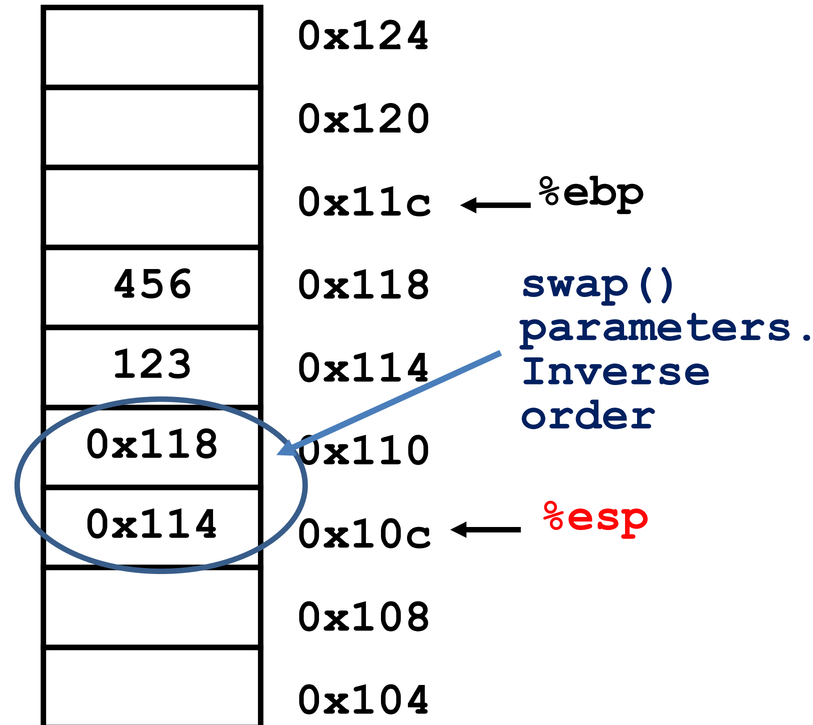



```
…
subl     $8, %esp
movl    $123, -8(%ebp)
movl     $456, -4(%ebp)
leal     -4(%ebp), %eax
pushl   %eax
leal     -8(%ebp), %eax
pushl   %eax
call     swap
...
```

Ένα απλό παράδειγμα

```
int main() {
  int one = 123, two = 456;
  swap(&one, &two);
  ...
}
```

```
…
subl     $8, %esp
movl    $123, -8(%ebp)
movl     $456, -4(%ebp)
leal     -4(%ebp), %eax
pushl   %eax
leal     -8(%ebp), %eax
pushl   %eax
call     swap
...
```

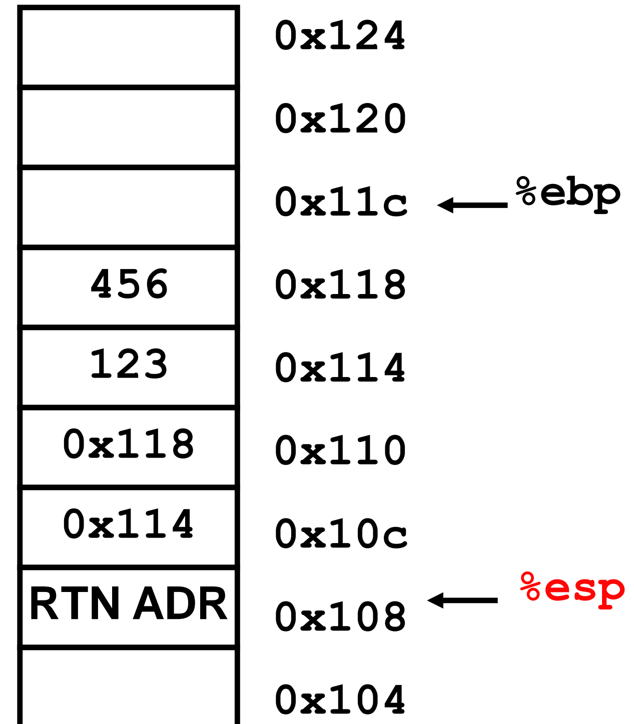


Key:
M[ESP - 4] = next_EIP
ESP = ESP - 4
EIP = swap

Ένα απλό παράδειγμα

```
int main() {
  int one = 123, two = 456;
  swap(&one, &two);
  ...
}
```

```
…
subl    $8, %esp
movl    $123, -8(%ebp)
movl    $456, -4(%ebp)
leal    -4(%ebp), %eax
pushl   %eax
leal    -8(%ebp), %eax
pushl   %eax
call    swap
...
```



Συνάρτηση *swap*

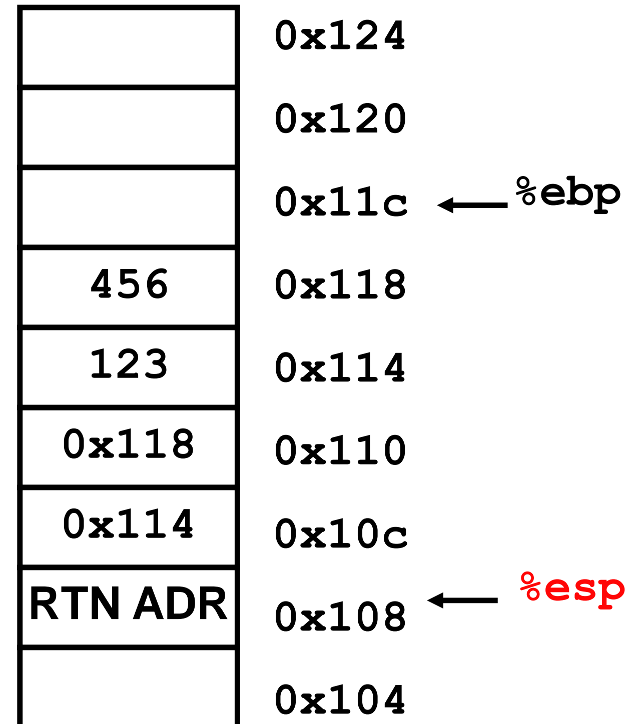
```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

```
swap:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx

    movl 12(%ebp),%ecx
    movl 8(%ebp),%edx
    movl (%ecx),%eax
    movl (%edx),%ebx
    movl %eax,(%edx)
    movl %ebx,(%ecx)

    movl -4(%ebp),%ebx
    movl %ebp,%esp
    popl %ebp
    ret
```

- Πρόλογος: pushl %ebp / movl %esp,%ebp / pushl %ebx
- Σώμα: movl 12(%ebp),%ecx … movl %ebx,(%ecx)
- Επίλογος: movl -4(%ebp),%ebx … ret

Πρόλογος *swap*

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

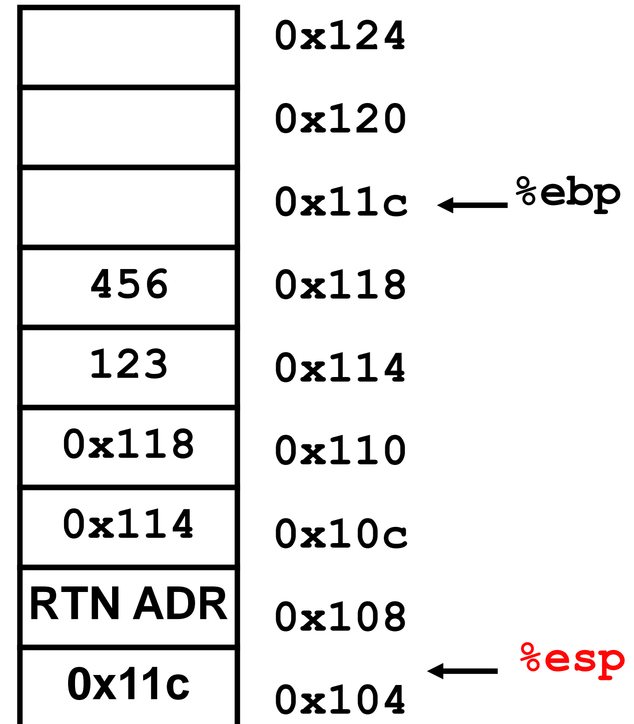



```
swap:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
```

Σώζουμε τον παλιό base pointer στην στοίβα

Πρόλογος *swap*

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```



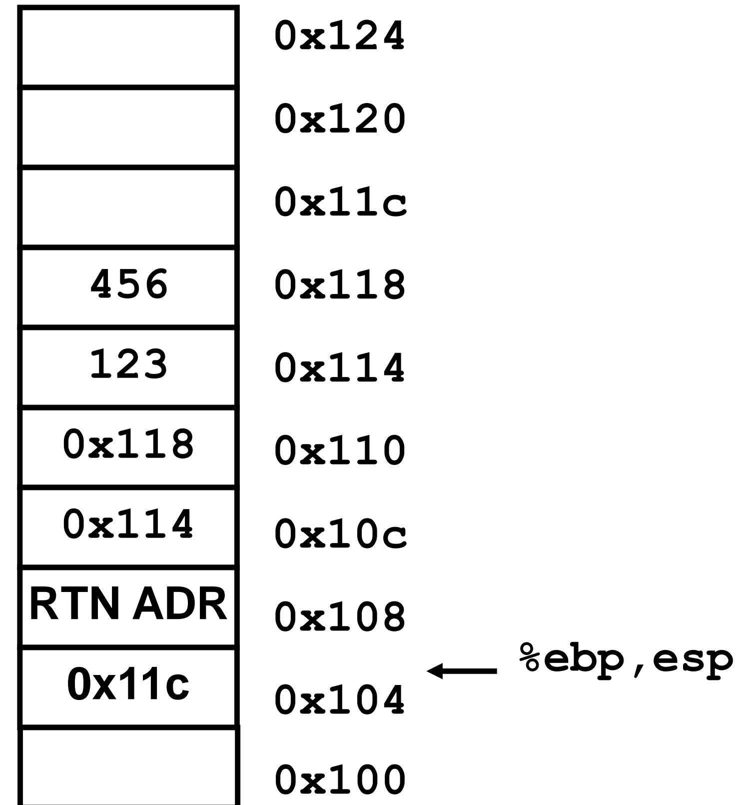

```
swap:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
```

Ο παλιός stack pointer γίνεται ο νέος base pointer

Πρόλογος *swap*

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

```
swap:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
```

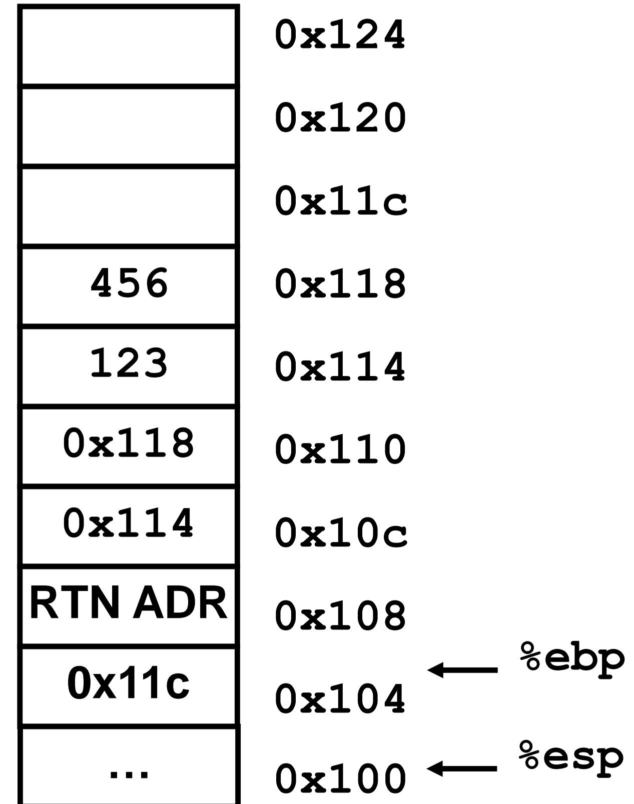


*Σώζουμε τον καταχωρητή **ebx** γιατί θα γίνει overwrite στην swap (callee saved).*

Πρόλογος *swap*

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

```
swap:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
```



Σώμα της *swap*

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

Register	Variable
%ecx	yp
%edx	xp
%eax	t1
%ebx	t0

Value	Address
456	0x118
123	0x114
0x118	0x110
0x114	0x10c
RTN ADR	0x108
0x11c	0x104 ← %ebp
...	0x100 ← %esp

```
movl 12(%ebp),%ecx    # ecx = yp
movl 8(%ebp),%edx     # edx = xp
movl (%ecx),%eax      # eax = *yp (t1)
movl (%edx),%ebx      # ebx = *xp (t0)
movl %eax,(%edx)      # *xp = eax
movl %ebx,(%ecx)      # *yp = ebx
```

Register	Value
%eax	
%edx	
%ecx	0x118
%ebx	

Σώμα της *swap*

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

Register	Variable
%ecx	yp
%edx	xp
%eax	t1
%ebx	t0

Value	Address
456	0x118
123	0x114
0x118	0x110
0x114	0x10c
RTN ADR	0x108
0x11c	0x104 ← %ebp
...	0x100 ← %esp

```
movl 12(%ebp),%ecx      # ecx = yp
movl 8(%ebp),%edx       # edx = xp
movl (%ecx),%eax        # eax = *yp (t1)
movl (%edx),%ebx        # ebx = *xp (t0)
movl %eax,(%edx)        # *xp = eax
movl %ebx,(%ecx)        # *yp = ebx
```

Register	Value
%eax	
%edx	0x114
%ecx	0x118
%ebx	

Σώμα της *swap*

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

Register	Variable
%ecx	yp
%edx	xp
%eax	t1
%ebx	t0

Value	Address
456	0x118
123	0x114
0x118	0x110
0x114	0x10c
RTN ADR	0x108
0x11c	0x104 ← %ebp
...	0x100 ← %esp

```
movl 12(%ebp),%ecx      # ecx = yp
movl 8(%ebp),%edx       # edx = xp
movl (%ecx),%eax        # eax = *yp (t1)
movl (%edx),%ebx        # ebx = *xp (t0)
movl %eax,(%edx)        # *xp = eax
movl %ebx,(%ecx)        # *yp = ebx
```

Register	Value
%eax	456
%edx	0x114
%ecx	0x118
%ebx	

Σώμα της *swap*

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

Register	Variable
%ecx	yp
%edx	xp
%eax	t1
%ebx	t0

Value	Address
456	0x118
123	0x114
0x118	0x110
0x114	0x10c
RTN ADR	0x108
0x11c	0x104 ← %ebp
...	0x100 ← %esp

```
movl 12(%ebp),%ecx     # ecx = yp
movl 8(%ebp),%edx      # edx = xp
movl (%ecx),%eax       # eax = *yp (t1)
movl (%edx),%ebx       # ebx = *xp (t0)
movl %eax,(%edx)       # *xp = eax
movl %ebx,(%ecx)       # *yp = ebx
```

Register	Value
%eax	456
%edx	0x114
%ecx	0x118
%ebx	123

Σώμα της *swap*

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

Register	Variable
%ecx	yp
%edx	xp
%eax	t1
%ebx	t0

Value	Address
456	0x118
456	0x114
0x118	0x110
0x114	0x10c
RTN ADR	0x108
0x11c	0x104 ← %ebp
...	0x100 ← %esp

```
movl 12(%ebp),%ecx     # ecx = yp
movl 8(%ebp),%edx      # edx = xp
movl (%ecx),%eax       # eax = *yp (t1)
movl (%edx),%ebx       # ebx = *xp (t0)
movl %eax,(%edx)       # *xp = eax
movl %ebx,(%ecx)       # *yp = ebx
```

Register	Value
%eax	456
%edx	0x114
%ecx	0x118
%ebx	123

Σώμα της *swap*

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

Register	Variable
%ecx	yp
%edx	xp
%eax	t1
%ebx	t0

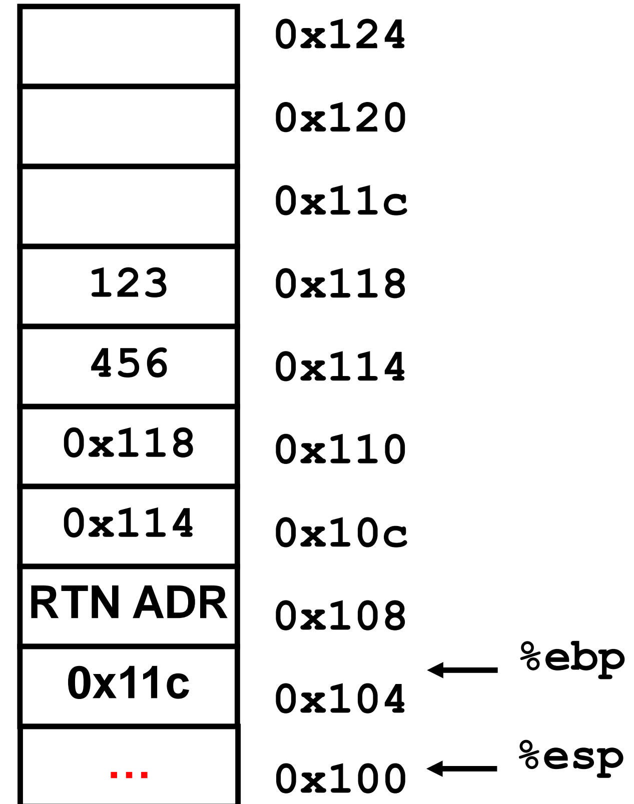
Value	Address
123	0x118
456	0x114
0x118	0x110
0x114	0x10c
RTN ADR	0x108
0x11c	0x104 ← %ebp
...	0x100 ← %esp

```
movl 12(%ebp),%ecx    # ecx = yp
movl 8(%ebp),%edx     # edx = xp
movl (%ecx),%eax      # eax = *yp (t1)
movl (%edx),%ebx      # ebx = *xp (t0)
movl %eax,(%edx)      # *xp = eax
movl %ebx,(%ecx)      # *yp = ebx
```

Register	Value
%eax	456
%edx	0x114
%ecx	0x118
%ebx	123

Επίλογος της *swap*

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```



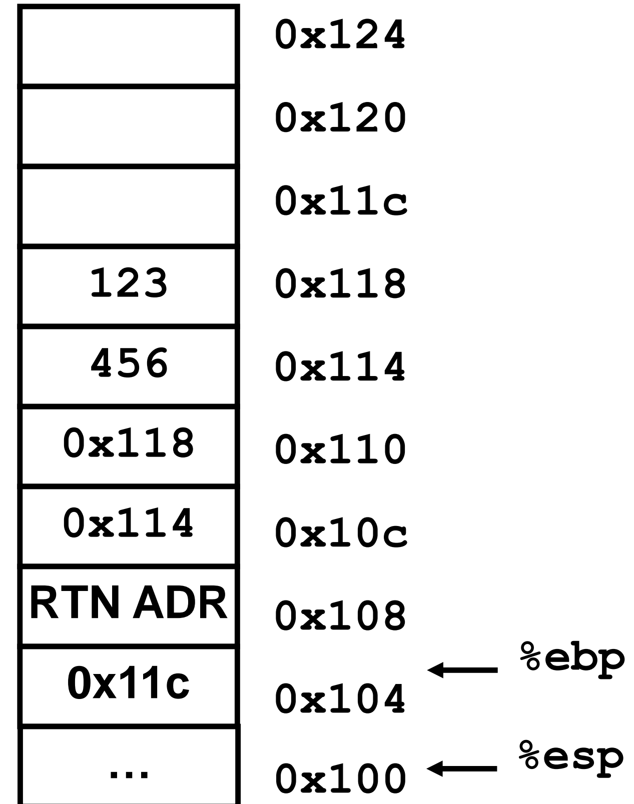

```
movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
```

*Επαναφέρουμε τον καταχωρητή **ebx***

Επίλογος της *swap*

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

```
movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
```

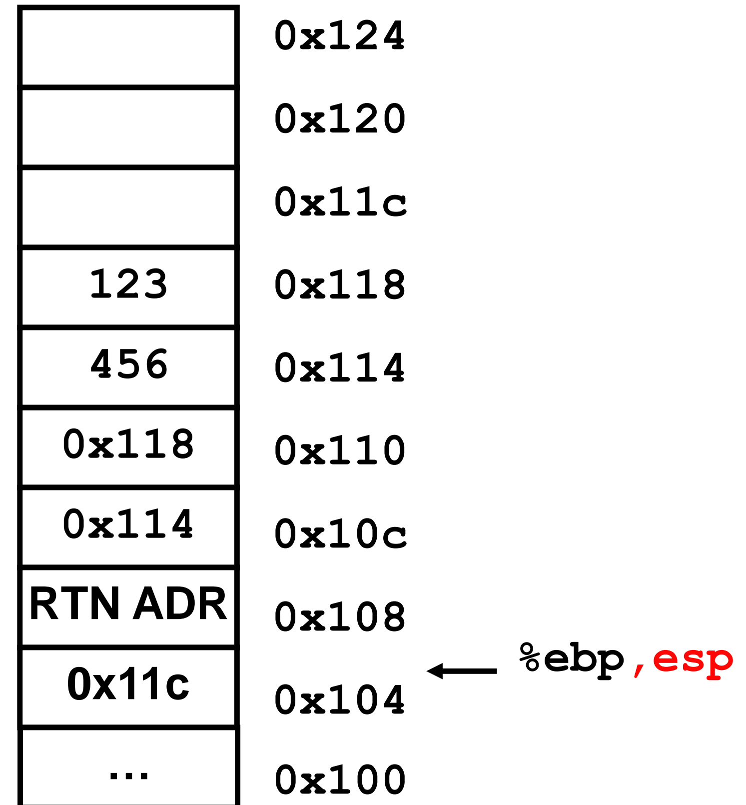


esp ← *ebp*

Επίλογος της *swap*

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

```
movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
```

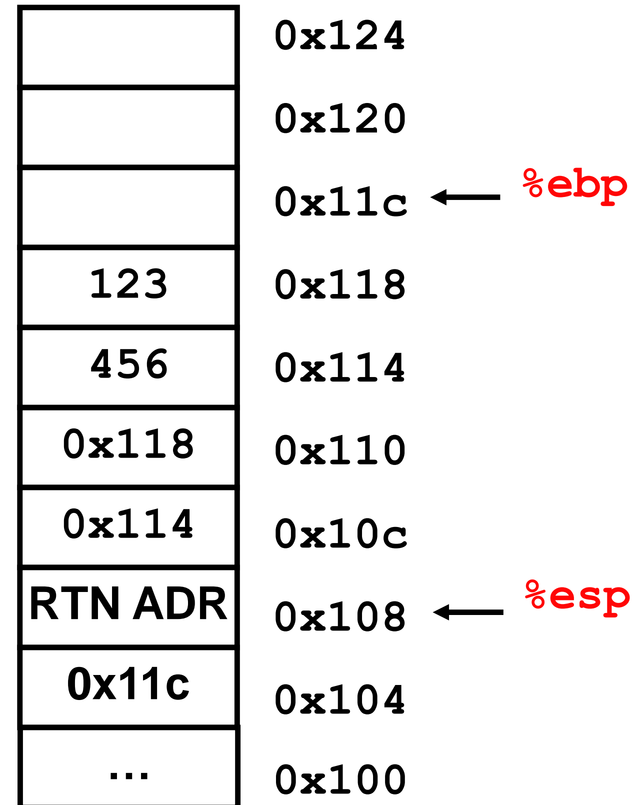


*Επαναφέρουμε τον καταχωρητή **ebp** στην παλιά του τιμή*

Επίλογος της *swap*

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

```
movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
```



Επιστροφή, και pop της διεύθυνσης επιστροφής από την στοίβα

Πράξεις με την μνήμη

- Οι περισσότερες εντολές μπορούν να επεξεργάζονται δεδομένα που είναι στην μνήμη ΧΩΡΙΣ να χρειάζεται να φορτωθούν πρώτα τα δεδομένα σε καταχωρητές
  - **addl -8(%ebp), %eax** # EAX = EAX + M[EBP - 8]
  - **incl -8(%ebp)** # M[EBP - 8] = M[EBP - 8] + 1

- Πιο πολύπλοκοι τρόποι διευθυνσιοδότησης:
  - **off(Rb,Ri,S)** # Mem[Reg[Rb]+S*Reg[Ri]+ off]
    - off: Offset μεγέθους από 1 μέχρι 4 bytes
    - Rb: Base register: Οποιοσδήποτε από τους 8 ακεραίους καταχωρητές
    - Ri: Index register: Οποιοσδήποτε, εκτός του `%esp`
    - S: Scale: 1, 2, 4, ή 8
  - Πολύ χρήσιμος για προσπέλαση πινάκων και structs

Παραδείγματα υπολογισμού διεύθυνσης

%edx	**0xf000**
%ecx	**0x100**

Expression	Computation	Address
0x8(%edx)	0xf000 + 0x8	0xf008
(%edx,%ecx)	0xf000 + 0x100	0xf100
(%edx,%ecx,4)	0xf000 + 4*0x100	0xf400
0x80(,%edx,2)	2*0xf000 + 0x80	0x1e080

Διακλαδώσεις

- Ο έλεγχος ροής ενός προγράμματος αποτελείται από δύο βήματα :
  - Θέτουμε ένα condition flag στον καταχωρητή EFLAGS
    - Αυτό γίνεται σαν παρενέργεια των περισσότερων αριθμητικών εντολών ή μέσω της εντολής **cmp**
  - Διακλαδώνουμε βασιζόμενοι στο condition flag

- Η συνήθης χρήση είναι μέσω της εντολής **cmp**
  - Είναι ακριβώς σαν την **sub**, χωρίς να γράφει το αποτέλεσμα

```
cmp     8(%ebx), %eax     # set flags based on (EAX - M[EBX + 8])
jg      branch_target     # taken if (EAX > M[EBX + 8])
```

Παράδειγμα με διακλαδώσεις

```
int sum(int n) {
  int i, sum = 0;
  for (i = 1 ; i <= n ; ++ i) {
    sum += i;
  }
  return sum;
}
```

Παράδειγμα με διακλαδώσεις

```c
int sum(int n) {
  int i, sum = 0;
  for (i = 1 ; i <= n ; ++ i) {
    sum += i;
  }
  return sum;
}
```

```
sum:    pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %ecx # n (was argument)
        movl    $1, %edx      # i = 1
        xorl    %eax, %eax    # sum = 0
        cmpl    %ecx, %edx    # (i ? n), sets cond. codes
        jg      .L8           # branch if (i > n)
.L6:
        addl    %edx, %eax    # sum += i
        incl    %edx          # i += 1
        cmpl    %ecx, %edx    # (i ? n)
        jle     .L6           # branch if (i <= n)
.L8:
```

Εντολές μεταβλητού μήκους

```
08048344 <sum>:
 8048344:       55                      push   %ebp
 8048345:       89 e5                   mov    %esp,%ebp
 8048347:       8b 4d 08                mov    0x8(%ebp),%ecx
 804834a:       ba 01 00 00 00          mov    $0x1,%edx
 804834f:       31 c0                   xor    %eax,%eax
 8048351:       39 ca                   cmp    %ecx,%edx
 8048353:       7f 0a                   jg     804835f
 8048355:       8d 76 00                lea    0x0(%esi),%esi
   ...
 804835f:       c9                      leave
 8048360:       c3                      ret
```

- Οι εντολές έχουν μήκος από 1 μέχρι και 17 17 bytes
  - Οι εντολές που χρησιμοποιούνται πολύ συχνά είναι μικρότερες (γιατί;)
    - Σε γενικές γραμμές, ο ίδιος πηγαίος κώδικας έχει μικρότερο μέγεθος στον x86 παρά στον MIPS assembly
    - Πιο δύσκολη η αποκωδικοποίηση του x86 προγράμματος από το hardware

Υλοποίηση του IA-32/64

- Το μεταβλητό μέγεθος των εντολών κάνει δύσκολο το IF, ID και EX
  - Για αυτό, πριν εκτελεσθούν, οι εντολές x86 “σπάνε” από το HW σε μικρότερες RISC-like micro-operations
  - Η εκτέλεση των micro-ops γίνεται όπως και των εντολών RISC