

Τελικός κώδικας

(i)

- Από θεωρητικής άποψης, το πρόβλημα της κατασκευής βέλτιστου τελικού κώδικα δεν έχει λύση (undecidable)
- Μορφές τελικού κώδικα:
 - Γλώσσα μηχανής σε απόλυτη μορφή (absolute)
 - Γλώσσα μηχανής σε επανατοποθετήσιμη και διασυνδέσιμη μορφή (relocatable, linkable)
 - Συμβολική γλώσσα (assembly)
 - Άλλη γλώσσα χαμηλού επιπέδου

Τελικός κώδικας

(ii)

- Επιμέρους προβλήματα:
 - Επιλογή εντολών
 - ⇒ Πώς μεταφράζεται κάθε εντολή του ενδιαμέσου κώδικα
 - ⇒ Πώς μεταφράζονται ακολουθίες τέτοιων εντολών
 - Διαχείριση της μνήμης στο χρόνο εκτέλεσης
 - ⇒ Πού αποθηκεύονται τα δεδομένα
 - ⇒ Πώς γίνεται η επικοινωνία ανάμεσα στις δομικές μονάδες

Τελικός υπολογιστής

(i)

- Χαρακτηριστικά:
 - Επεξεργαστής: Intel 8086
 - Λειτουργικό σύστημα: MS-DOS
 - Μοντέλο μνήμης: COM / tiny
 - ⇒ Συνολική μνήμη ≤ 64 K
 - ⇒ Οργάνωση σε ένα segment
 - ⇒ Αρχική διεύθυνση του προγράμματος η $100h$
 - Συμβολική γλώσσα: συμβατή με το συμβολομεταφραστή MASM (Microsoft macro assembler)

Τελικός υπολογιστής

(ii)

- Καταχωρητές, μεγέθους 16 bit
 - Γενικής φύσης: ax, bx, cx, dx
⇒ σε ζεύγη των 8 bit: ah, al, κ.λπ.
 - Καταχωρητές δείκτες: sp (δείκτης στοίβας) και bp (δείκτης βάσης)
 - Καταχωρητές αναφοράς: si και di
 - Καταχωρητές τμημάτων: cs (code), ds (data), ss (stack) και es (extra)
 - Ειδικό καταχωρητές: ip (instruction pointer) και καταχωρητής σημαίων (flags)

Τελικός υπολογιστής

(iii)

- Διευθύνσεις:

$$address = segment * 16 + offset$$

- Μορφή εντολής:

[*label*] *opcode* [*operand*₁ [, *operand*₂]]

Τελικός υπολογιστής

(iv)

■ Εντολές:

- Μεταφοράς: `mov`, `lea`
- Αριθμητικών πράξεων: `add`, `sub`, `neg`, `imul`, `idiv`, `cmp`, `cwd`
- Λογικών πράξεων: `and`, `or`, `xor`, `not`, `test`
- Άλματος: `jmp`, `jz`, `jnz`, `jl`, `jle`, `jc`, `jge`
- Διαχείρισης στοίβας: `push`, `pop`
- Υποπρογραμμάτων: `call`, `ret`
- Πράξεων κινητής υποδιαστολής (x87 FPU)

Εντολές μεταφοράς

- *mov destination, source* (move)

```
mov ax, 42
mov ax, bx
mov ax, [1000h]
mov ax, [si]
mov ax, [si + 6]
mov ax, [bp + 6]
mov ax, [si + bp + 6]
```

- *lea destination, source* (load effective address)

- Καθορισμός μεγέθους δεδομένων

```
mov ax, word ptr [bp + 6]
mov al, byte ptr [bp + 6]
```

Αριθμητικές πράξεις

■ `add op1, op2`

$op_1 := op_1 + op_2$

■ `sub op1, op2`

$op_1 := op_1 - op_2$

■ `neg op`

$op := -op$

■ `imul op`

$(dx, ax) := ax * op$

■ `idiv op`

$ax := (dx, ax) \text{ div } op$

$dx := (dx, ax) \text{ mod } op$

■ `cwd`

επέκταση προσήμου του `ax` στον `dx`

■ `cmp op1, op2`

σύγκρινε τα `op1` και `op2`
ενημέρωσε τις σημαίες



Λογικές πράξεις

- `and op1, op2`
- `or op1, op2`
- `not op`
- `xor op1, op2`
- `test op1, op2`

`op1 := op1 and op2`

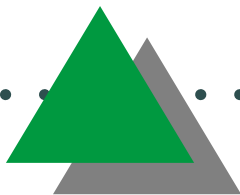
`op1 := op1 or op2`

`op := not op`

`op1 := op1 xor op2`

`op1 and op2`

ενημέρωσε τις σημαίες



Εντολές άλματος

- *jmp address* χωρίς συνθήκη
- *jz address* ή *je address* μηδέν / ίσο
- *jnz address* ή *jne address* όχι μηδέν / διάφορο
- *jl address* μικρότερο
- *jle address* μικρότερο ή ίσο
- *jg address* μεγαλύτερο
- *jge address* μεγαλύτερο ή ίσο

Εντολές στοίβας

■ *push op*

πρόσθεση στη στοίβα
 $sp := sp - 2, [sp] := op$

■ *pop op*

αφαίρεση από τη στοίβα
 $op := [sp], sp := sp + 2$

⇒ Η στοίβα αυξάνει προς τα κάτω, δηλαδή προς μικρότερες διευθύνσεις



Εντολές υποπρογραμμάτων

■ *call address*

κλήση

$sp := sp - 2, [sp] := ip, ip := address$

■ *ret*

επιστροφή

$ip := [sp], sp := sp + 2$

⇒ Η τιμή του *ip* που τοποθετείται στη στοίβα από την *call* είναι η διεύθυνση της εντολής που ακολουθεί την *call*



Εντολές x87 FPU

(i)

⇒ Ειδική στοίβα 8 καταχωρητών: ST(0), ... ST(7)

■ *fld source* (load real & push)

```
fld tbyte ptr @real1
```

■ *fild source* (load integer & push)

```
fild word ptr [bp - 2]
```

■ *fstp destination* (pop & store real)

```
fld tbyte ptr [bp - 10]
```

■ *fistp destination* (pop & store integer)

```
fild word ptr [bp - 2]
```

Εντολές x87 FPU

(ii)

- `faddp ST(1), ST(0)` $ST(1) := ST(1) + ST(0) \ \& \ pop$
- `fsubp ST(1), ST(0)` $ST(1) := ST(1) - ST(0) \ \& \ pop$
- `fmulp ST(1), ST(0)` $ST(1) := ST(1) * ST(0) \ \& \ pop$
- `fdivp ST(1), ST(0)` $ST(1) := ST(1) / ST(0) \ \& \ pop$
- `fchs` $ST(0) := -ST(0)$
- `fcompp` $ST(1) \begin{matrix} \geq \\ \leq \end{matrix} ST(0) \ \& \ pop \ both$
- `fstsw destination` (store x87 FPU flags)

```
fstsw ax
```

```
fstsw word ptr [bp - 2]
```

Διαχείριση μνήμης

(i)

- Δομή ενοτήτων (block structure)
 - Μη τοπικά δεδομένα
- Εγγράφημα δραστηριοποίησης (activation record)
 - Παράμετροι
 - Αποτέλεσμα
 - Πληροφορίες κατάστασης μηχανής
 - Τοπικές μεταβλητές
 - Προσωρινές μεταβλητές

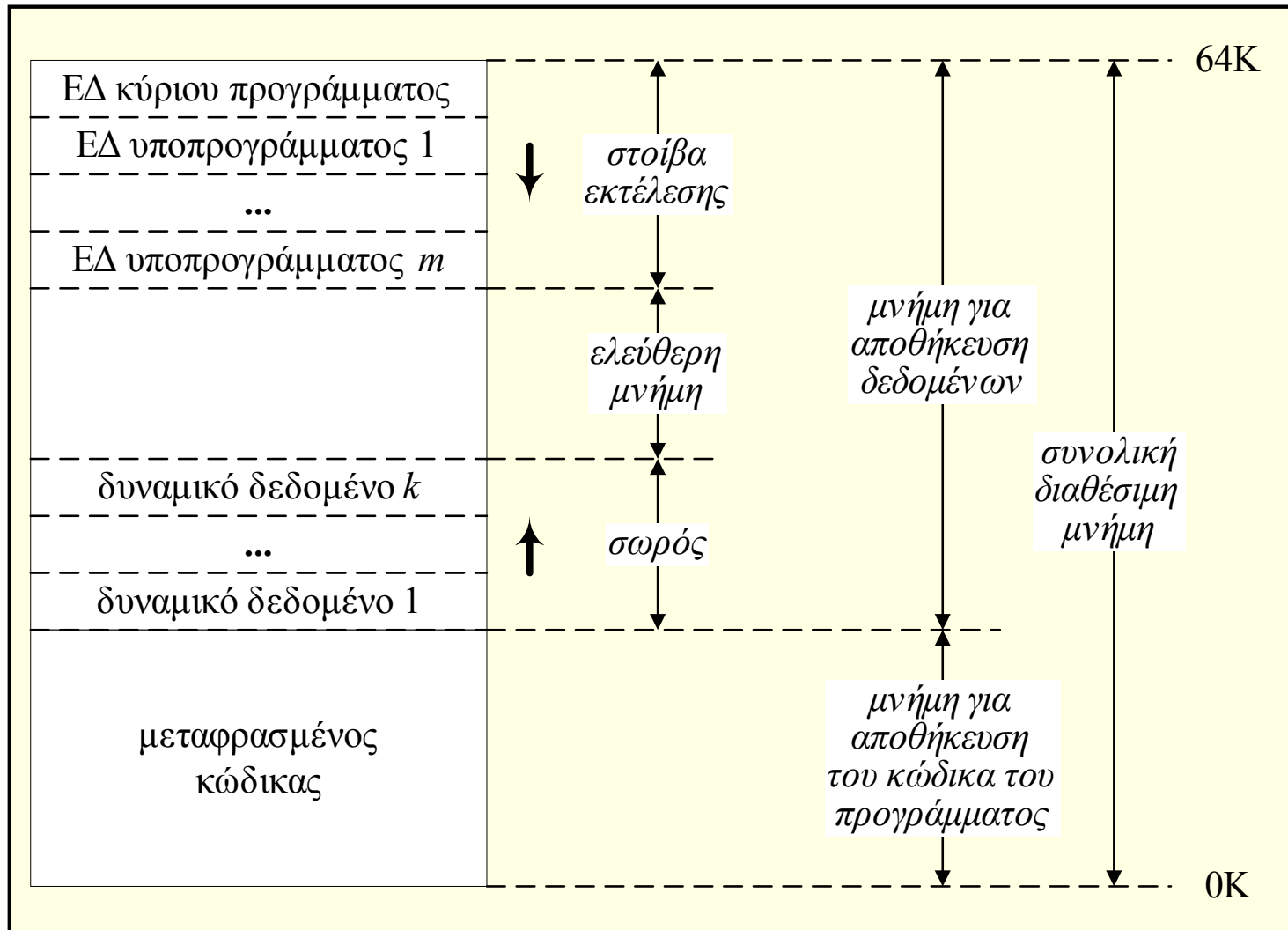
Διαχείριση μνήμης

(ii)

	Παράμετρος 1		Παράμετρος 1	Παράμετροι	αρχή
	Παράμετρος 2		Παράμετρος 2		
...		
br+8	Παράμετρος n		Παράμετρος n		
br+6	Διεύθυνση αποτελέσματος		Διεύθυνση αποτελέσματος	Σταθερό Τμήμα	βάση
br+4	Σύνδεσμος προσπέλασης		Διεύθυνση επιστροφής		
br+2	Διεύθυνση επιστροφής		Προηγούμενο display		
br	Προηγούμενο br		Προηγούμενο br		
br-2	Τοπική μεταβλητή 1		Τοπική μεταβλητή 1	Τοπικές μεταβλητές	τέλος
br-4	Τοπική μεταβλητή 2		Τοπική μεταβλητή 2		
...		
	Τοπική μεταβλητή m		Τοπική μεταβλητή m		
	Προσωρινή μεταβλητή 1		Προσωρινή μεταβλητή 1	Προσωρινές μεταβλητές	τέλος
	Προσωρινή μεταβλητή 2		Προσωρινή μεταβλητή 2		
		
	Προσωρινή μεταβλητή k		Προσωρινή μεταβλητή k		
α) Σύνδεσμοι προσπέλασης			β) Πίνακας δεικτών		

Διαχείριση μνήμης

(iii)



Προσπέλαση ονομάτων

- Τοπικά: $[bp + offset]$
- Μη τοπικά: $[si + offset]$
 - ⇒ ο si πρέπει να δείχνει στη βάση του εγγραφήματος δραστηριοποίησης όπου τα δεδομένα είναι τοπικά
- Το πρόβλημα ανάγεται στον εντοπισμό του αντίστοιχου εγγραφήματος δραστηριοποίησης
- Λύσεις που βασίζονται στο βάθος φωλιάσματος:
 - Σύνδεσμοι προσπέλασης (access links)
 - Πίνακες δεικτών (link tables / displays)

Σύνδεσμοι προσπέλασης (i)

■ Αρχή λειτουργίας

- Έστω ότι η δομική μονάδα p βρίσκεται φωλιασμένη μέσα στη δομική μονάδα q

⇒ Στο ΕΔ της p τοποθετείται ένα σύνδεσμος προς τη βάση του ΕΔ της **πιο πρόσφατης κλήσης** της q

- Κατά την κλήση υποπρογραμμάτων, απαιτείται τελικός κώδικας για την ενημέρωση των συνδέσμων προσπέλασης

Σύνδεσμοι προσπέλασης (ii)

■ Τρόπος χρήσης

- Έστω ότι ζητείται το δεδομένο a που είναι τοπικό σε μια δομική μονάδα με βάθος φωλιάσματος n_a

- Έστω ότι βρισκόμαστε σε μια δομική μονάδα p με βάθος φωλιάσματος $n_p \geq n_a$

⇒ Ακολουθούμε $n_p - n_a$ συνδέσμους προσπέλασης

- Κατά την προσπέλαση ονομάτων, απαιτείται τελικός κώδικας για την υλοποίηση των παραπάνω

Πέρασμα παραμέτρων

- Κλήση κατ' αξία (call by value)
- Κλήση κατ' αναφορά (call by reference)
- Κλήση κατ' όνομα (call by name)
- Κλήση κατ' ανάγκη (call by need)
- Κλήση κατ' αξία και αποτέλεσμα (call by value-result)

⇒ Τρόπος υλοποίησης καθενός

Δέσμευση καταχωρητών (i)

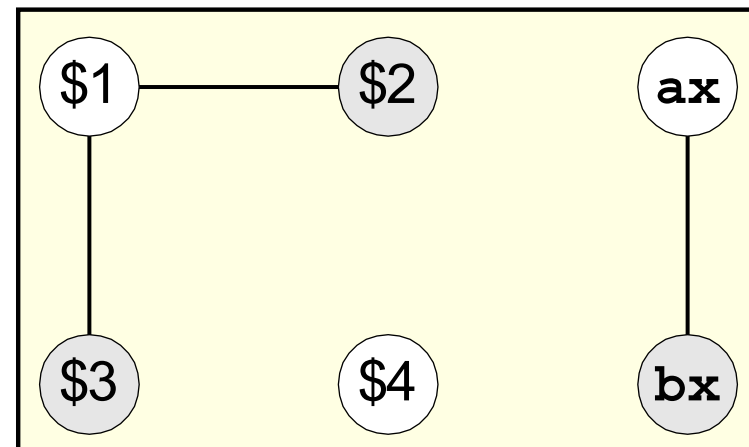
- Πρόβλημα 1: επιλογή τελουμένων που θα αποθηκευτούν σε καταχωρητές
- Πρόβλημα 2: επιλογή καταχωρητών όπου θα αποθηκευτούν τα τελούμενα
- Το πρόβλημα της βέλτιστης χρήσης καταχωρητών είναι **NP-complete**, ακόμα και χωρίς ειδικούς περιορισμούς
- Η λύση του ανάγεται στην κατασκευή του **γράφου αλληλεπιδράσεων** μεταξύ των μεταβλητών και στο **χρωματισμό** αυτού με τόσα χρώματα όσοι οι καταχωρητές

Δέσμευση καταχωρητών (ii)

■ Παράδειγμα:

$d := b*b - 4*a*c$

- 1: *, b, b, \$1
- 2: *, 4, a, \$2
- 3: *, \$2, c, \$3
- 4: -, \$1, \$3, \$4
- 5: :=, \$4, -, d



Επιλογή εντολών

(i)

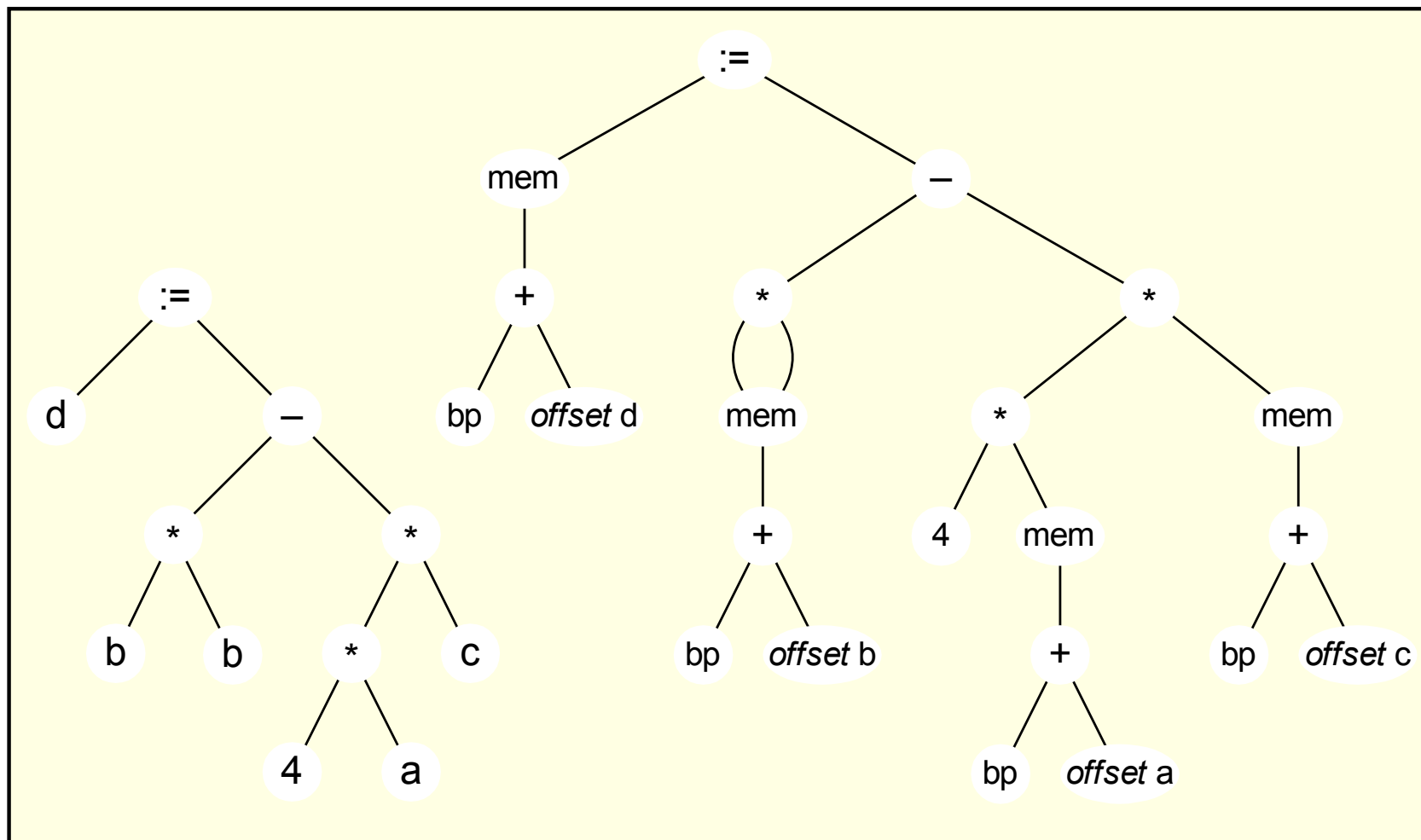
- Απλή αλλά κακή προσέγγιση: ενιαίο σχήμα παραγωγής τελικού κώδικα για κάθε δομή του ενδιάμεσου κώδικα
- Καλύτερη προσέγγιση: πλακόστρωση (tiling)
 - τεμαχισμός του ενδιάμεσου κώδικα σε τμήματα
 - κάθε τμήμα αντιστοιχεί σε μια εντολή
- Βέλτιστη αλλά χρονοβόρα προσέγγιση: δυναμικός προγραμματισμός (dynamic programming)

Επιλογή εντολών

(ii)

■ Παράδειγμα:

$d := b*b - 4*a*c$

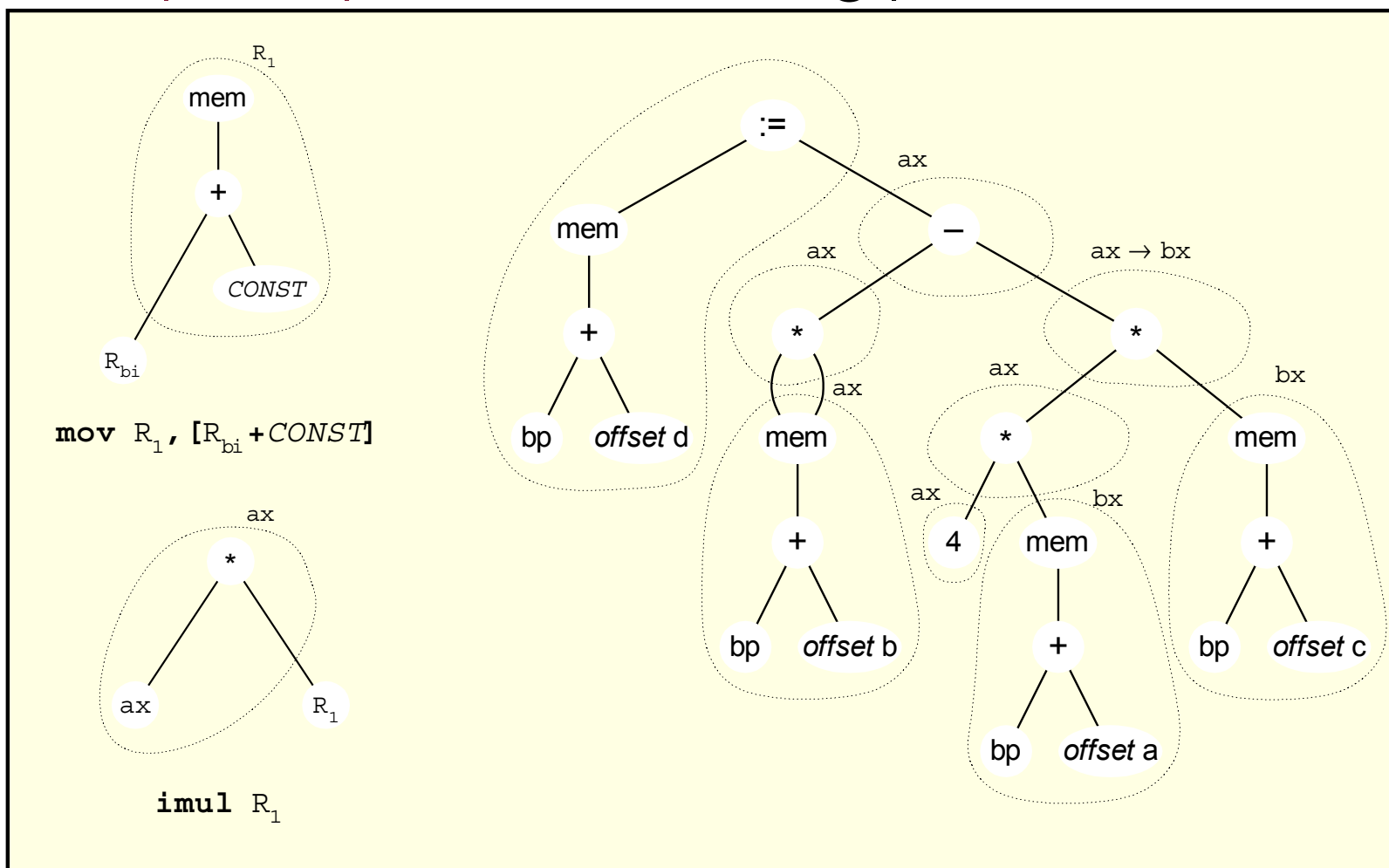


Επιλογή εντολών

(iii)

■ Παράδειγμα:

tiling με εντολές του 8086

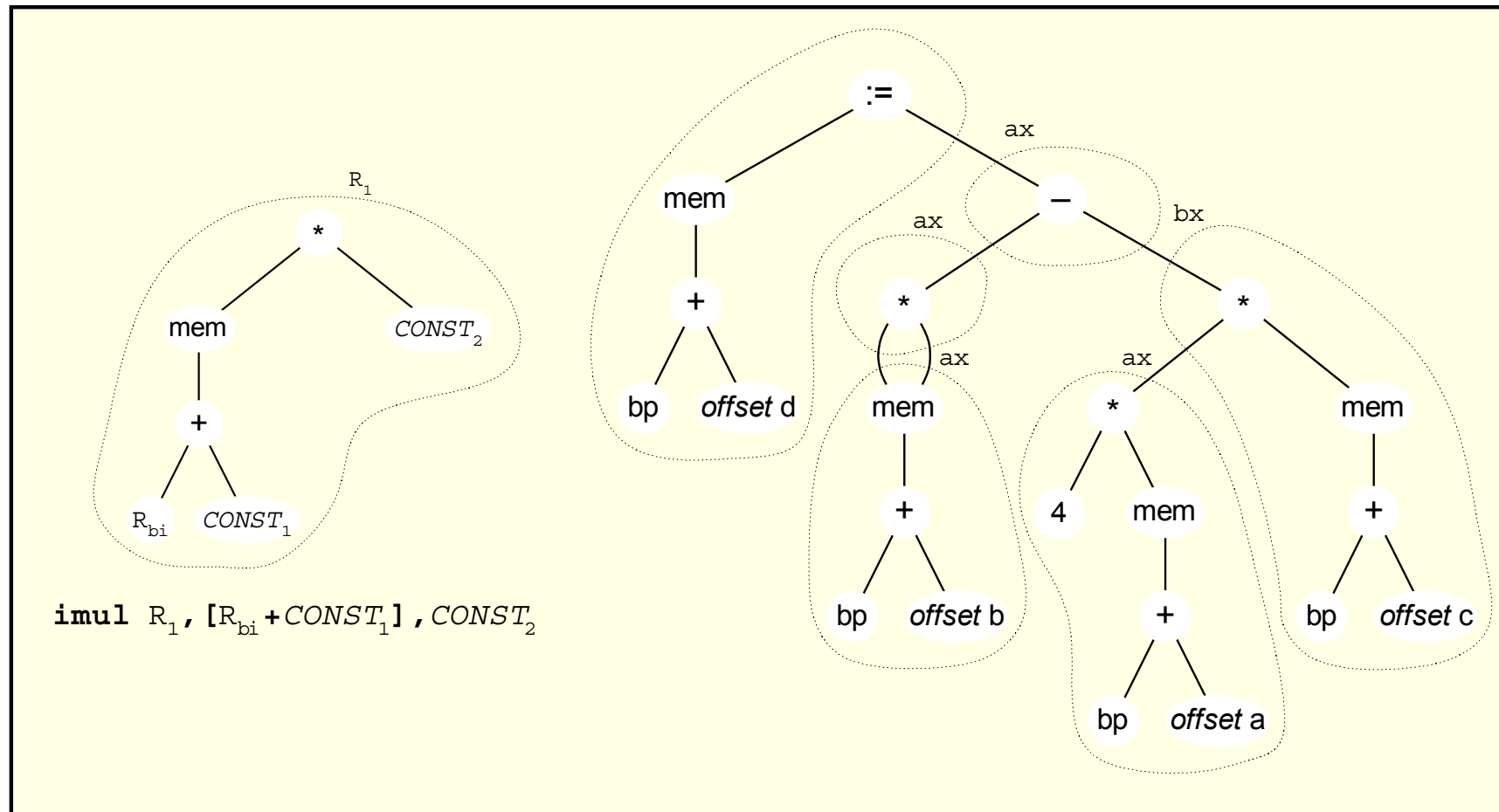


Επιλογή εντολών

(iv)

■ Παράδειγμα:

tiling με εντολές του 80386



Το τελικό πρόγραμμα

(i)

■ Σκελετός:

```
xseg segment public 'code'
    assume cs : xseg, ds : xseg, ss : xseg
    org     100h
main proc near
    call   near ptr program
    mov    ax, 4C00h
    int    21h
main endp
```

... τελικός κώδικας που παράγεται ...

```
xseg ends
    end    main
```

Το τελικό πρόγραμμα

(ii)

- Βιβλιοθήκη χρόνου εκτέλεσης (run-time library)

```
extrn function : proc
```

- Σταθερές συμβολοσειρές και κινητής υποδιαστολής

```
@str1  db 'this is'  
        db 10  
        db 'an example'  
        db 0
```

```
@real1 dt 1e-10
```

```
@real2 dt 2.0
```

Βοηθητικές ρουτίνες

(i)

- `getAR(a)` (φόρτωση διεύθυνσης ΕΔ)

```
mov si, word ptr [bp + 4]
mov si, word ptr [si + 4]
...
mov si, word ptr [si + 4]
```

} ($n_{\text{cur}} - n_a - 1$ φορές)

- `updateAL()` (ενημέρωση συνδέσμων προσπέλασης)

(α) `push bp` αν $n_p < n_x$

(β) `push word ptr [bp + 4]` αν $n_p = n_x$

(γ) `mov si, word ptr [bp + 4]` αν $n_p > n_x$

```
mov si, word ptr [si + 4]
...
mov si, word ptr [si + 4]
push word ptr [si + 4]
```

} ($n_p - n_x - 1$ φορές)

Βοηθητικές ρουτίνες

(ii)

■ `load(R, a)`

(φόρτωση τελουμένου)

	Είδος του <code>a</code>	Κώδικας που παράγεται
(α1)	αριθμητική σταθερά	<code>mov R, a</code>
(α2)	λογική σταθερά <code>true</code>	<code>mov R, 1</code>
(α3)	λογική σταθερά <code>false</code>	<code>mov R, 0</code>
(α4)	σταθερά χαρακτήρα	<code>mov R, ASCII(a)</code>
(α6)	σταθερά <code>nil</code>	<code>mov R, 0</code>
(β1)	τοπική οντότητα: μεταβλητή, παράμετρος κατ' αξία, ή προσωρινή μεταβλητή	<code>mov R, size ptr [bp + offset]</code>
(β2)	τοπική οντότητα: παράμετρος κατ' αναφορά	<code>mov si, word ptr [bp + offset]</code> <code>mov R, size ptr [si]</code>

Βοηθητικές ρουτίνες

(iii)

■ `load(R, a)`

(φόρτωση τελουμένου)

	Είδος του <i>a</i>	Κώδικας που παράγεται
(γ1)	μη τοπική οντότητα: μεταβλητή, παράμετρος κατ' αξία, ή προσωρινή μεταβλητή	<code>getAR(a)</code> <code>mov R, size ptr [si + offset]</code>
(γ2)	μη τοπική οντότητα: παράμετρος κατ' αναφορά	<code>getAR(a)</code> <code>mov si, word ptr [si + offset]</code> <code>mov R, size ptr [si]</code>
(δ)	$[x]$	<code>load(di, x)</code> <code>mov R, size ptr [di]</code>
(ε)	$\{x\}$	<code>loadAddr(R, x)</code>

Βοηθητικές ρουτίνες

(iv)

- `loadAddr(R, a)` (φόρτωση διεύθυνσης τελουμένου)

	Είδος του <i>a</i>	Κώδικας που παράγεται
(α5)	σταθερή συμβολοσειρά	<code>lea R, byte ptr a</code>
(β1)	τοπική οντότητα: παράμετρος κατ' αξία, ή προσωρινή μεταβλητή	<code>lea R, size ptr [bp + offset]</code>
(β2)	τοπική οντότητα: παράμετρος κατ' αναφορά	<code>mov R, word ptr [bp + offset]</code>
(γ1)	μη τοπική οντότητα: παράμετρος κατ' αξία, ή προσωρινή μεταβλητή	<code>getAR(a)</code> <code>lea R, size ptr [si + offset]</code>
(γ2)	μη τοπική οντότητα: παράμετρος κατ' αναφορά	<code>getAR(a)</code> <code>mov R, word ptr [si + offset]</code>
(δ)	$[x]$	<code>load(R, x)</code>

Βοηθητικές ρουτίνες

(v)

- Παρόμοια υλοποίηση για τις:
 - `loadReal(a)`
 - `store(R, a)`
 - `storeReal(a)`
- Ρουτίνες για ετικέτες τελικού κώδικα:
 - `name(p)`
 - `endof(p)`
 - `label(n)`
 - `label(l)`

`_p_num`

`@p_num`

`@n`

`@p_num_l`

Παραγωγή κώδικα

(i)

- Τετράδα $:=, x, -, z$
load(R, x) loadReal(x)
store(R, z) storeReal(z)

- Τετράδα $array, x, y, z$
load(ax, y)
mov cx, size
imul cx
loadAddr(cx, x)
add ax, cx
store(ax, z)

Παραγωγή κώδικα

(ii)

- Τετράδες $+, x, y, z$ $-, x, y, z$

load(ax, x)

load(dx, y)

instr ax, dx

store(ax, z)

instr = add ή sub

loadReal(x)

loadReal(y)

finstr ST(1), ST(0)

storeReal(z)

finstr = faddp κ.λπ.

- Τετράδες $*, x, y, z$ $/, x, y, z$ $\%, x, y, z$

load(ax, x)

load(cx, y)

imul cx

store(ax, z)

load(ax, x)

cwd

load(cx, y)

idiv cx

store(ax, z)

load(ax, x)

cwd

load(cx, y)

idiv cx

store(dx, z)

Παραγωγή κώδικα

(iii)

- Τετράδες $=, x, y, z$ $<>, x, y, z$ $<, x, y, z$
 $>, x, y, z$ $<=, x, y, z$ $>=, x, y, z$

load(ax, x)

load(dx, y)

cmp ax, dx

instr label(z)

instr = je, jne, κ.λπ.

loadReal(x)

loadReal(y)

fcompp

fstsw ax

test ax, *value*

instr label(z)

value και *instr* από
Πίνακα 9.1 σελ. 249

Παραγωγή κώδικα

(iv)

- Τετράδα `ifb, x, -, z`
`load(a1, x)`
`or a1, a1`
`jnz label(z)`
- Τετράδα `jump, -, -, z`
`jmp label(z)`
- Τετράδα `jumppl, -, -, z`
`jmp label(z)`
- Τετράδα `label, -, -, z`

`label(z) :`

Παραγωγή κώδικα

(v)

- Τετράδα `unit, x, —, —`

```
name(x) proc near
        push bp
        mov  bp, sp
        sub  sp, size
```

- Τετράδα `endu, x, —, —`

```
endof(x) : mov  sp, bp
          pop  bp
          ret
name(x)   endp
```

Παραγωγή κώδικα

(vi)

- Τετράδα `call, -, -, z`

```
sub    sp, 2           αν z είναι διαδικασία
updateAL()
call   near ptr name(z)
add    sp, size + 4
```

- Τετράδα `ret, -, -, -`

```
jmp   endof(current)
```


Παραγωγή κώδικα

(vii)

- Τετράδα `par, x, y, -`
 - αν $y = V$ και x είναι 16 bit
`load(ax, x)`
`push ax`
 - αν $y = V$ και x είναι 8 bit
`load(al, x)`
`sub sp, 1`
`mov si, sp`
`mov byte ptr [si], al`

Παραγωγή κώδικα

(vii)

- Τετράδα `par, x, y, -` (συνέχεια)

- αν $y = V$ και x είναι 80 bit

```
loadReal(x)
sub    sp, 10
mov    si, sp
fstp   tbyte ptr [si]
```

- αν $y = R$ ή RET

```
loadAddr(si, x)
push  si
```