Προηγμένα Θέματα Θεωρητικής Πληροφορικής

Ανασκόπηση του μαθήματος -
Γέννηση τελικού κώδικα για RISC επεξεργαστές

Νικόλαος Καββαδίας
nkavv@uop.gr

02 Ιουνίου 2010

Σκιαγράφηση της διάλεξης

- Σύνοψη του μαθήματος
- Ενδεικτικά θέματα εξετάσεων
- Άλλα θέματα
- Η αρχιτεκτονική επεξεργαστή MIPS
- Γέννηση τελικού κώδικα για τον επεξεργαστή MIPS

Σύνοψη του μαθήματος (1)

1. Η οργάνωση του δομημένου μεταγλωττιστή
   - Η χρησιμότητα της ενδιάμεσης αναπαράστασης
   - Πρακτικοί μεταγλωττιστές
2. Γέννηση ενδιάμεσης αναπαράστασης
   - Αποδόμηση σύνθετων εκφράσεων της ANSI C
   - Βασικό μπλοκ - αναπαράσταση DAG
   - Κώδικας τριών διευθύνσεων (three address code)
   - Μεταγλώττιση πηγαίου κώδικα σε TAC
   - Γράφοι εξάρτησης δεδομένων
   - CFG και CDFG
   - Ορισμός της SSA
   - Κατασκευή SSA
3. Επιλογή κώδικα
   - Η έννοια της κοινής υποεκφράσεως
   - Μη βέλτιστη πλακόστρωση δένδρου ροής δεδομένων για την επιλογή κώδικα
   - Σχεδιασμός AST από κειμενική αναπαράσταση DFT

Σύνοψη του μαθήματος (2)

4. Καταμερισμός καταχωρητών
   - Διαστήματα ζωής: εξαγωγή από κώδικα TAC
   - Καθολικός καταμερισμός καταχωρητών
   - Χρωματισμός γράφου - Γράφοι παρεμβολής
   - Περιοχές ζωής - η διαφορά τους από τα διαστήματα ζωής
   - Ο αλγόριθμος του Chaitin και πως εφαρμόζεται
   - Ο αλγόριθμος γραμμικής σάρωσης: περιγραφή και εφαρμογή
5. Βελτιστοποιήσεις ανεξάρτητες από την αρχιτεκτονική
   - Ο βελτιστοποιητής στο πλαίσιο του δομημένου μεταγλωττιστή
   - Βασικές διαφορές μεταξύ βελτιστοποιήσεων υψηλού και χαμηλού επιπέδου - Παραδείγματα
   - Η εφαρμογή όλων των βαθμωτών βελτιστοποιήσεων
   - Σχεδιασμός δένδρου κυριαρχίας (όχι βέλτιστος αλγόριθμος)

Σύνοψη του μαθήματος (3)

6. Χρονοπρογραμματισμός κώδικα και βελτιστοποιήσεις εξαρτημένες από την αρχιτεκτονική
   - Εξαρτήσεις εντολών
   - Αρχές στατικού και δυναμικού χρονοπρογραμματισμού
   - Ο αλγόριθμος ASAP
   - Ο αλγόριθμος χρονοπρογραμματισμού λίστας
7. Βελτιστοποιήσεις (I)
   - Η διαδικασία της βελτιστοποίησης
   - Γενικευμένη δομή βρόχων και πεδίο επανάληψης
   - Loop unswitching, loop reversal
   - Strip mining, cycle shrinking
   - Loop tiling
   - Loop unrolling
8. Βελτιστοποιήσεις (II)
   - Scalar expansion, scalar replacement
   - Software pipelining και σύγκριση με loop unrolling
   - Modulo scheduling

Ενδεικτικά θέματα: Γενικά

- Να δοθεί το σχηματικό διάγραμμα του τυπικού σχεδιασμού ενός μεταγλωττιστή. Στη συνέχεια:
  α) Να ονομαστεί κάθε επιμέρους τμήμα του και να δοθεί σύντομη περιγραφή της λειτουργίας του.
  β) Ποια η λειτουργία του πίνακα συμβόλων (σύντομα)
  γ) Ποια τα πλεονεκτήματα της χρήσης ενδιάμεσης αναπαράστασης στο σχεδιασμό ενός επαναστοχεύσιμου μεταγλωττιστή· Να δοθεί αριθμητικό παράδειγμα για την περίπτωση μεταγλωττιστή ο οποίος δέχεται τις πηγαίες γλώσσες ANSI C, C++, και Pascal και παράγει κώδικα στις γλώσσες συμβολομεταφραστή για τις αρχιτεκτονικές x86, MIPS, ARM και PowerPC.

Ενδεικτικά θέματα: Ενδιάμεση αναπαράσταση (1)

- Να απαντηθούν τα παρακάτω ερωτήματα:
  α) Τι είναι βασικό μπλοκ σε έναν γράφο ροής ελέγχου και ποια τα χαρακτηριστικά του. Δώστε ένα παράδειγμα βασικού μπλοκ (μέχρι 7 εντολές) με κώδικα τριών διευθύνσεων (TAC).
  β) Τι είναι η μορφή Στατικής Απλής Ανάθεσης (SSA) και ποια η κύρια ιδιότητά της·
  γ) Να παραχθεί γράφος ροής ελέγχου με βασικά μπλοκ σε SSA μορφή για το παρακάτω τμήμα κώδικα.

```
i = 2;
j = 1;
do
{
  if (j < 7)
  {
    i = i + 1;
    j = j + i;
  }
  else
  {
    break;
  }
} while (i <= 10);
```

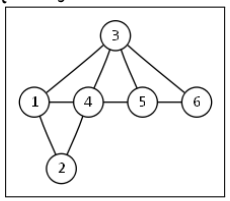

Ενδεικτικά θέματα: Ενδιάμεση αναπαράσταση (2)

- Να απαντηθούν τα παρακάτω:
  α) Ο παρακάτω κώδικας ANSI C περιγράφει έναν αλγόριθμο παραγοντοποίησης του ακεραίου n σε γινόμενο πρώτων αριθμών (prime factorization). Πρώτος αριθμός είναι αυτός που διαιρείται ακριβώς μόνο με τον εαυτό του και τη μονάδα. Να δοθεί ο γράφος ροής ελέγχου (CFG) σε non-SSA και σε SSA μορφή για τον αλγόριθμο.

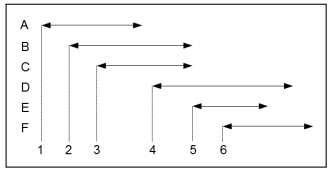
```
i = 2;
while (i <= n) {
  while ((n % i) == 0) {
    n = n / i;
    printf("%d * ", i);
  }
  i = i + 1;
}
```

Ενδεικτικά θέματα: Καταμερισμός καταχωρητών

- Να πραγματοποιηθεί καταμερισμός καταχωρητών:
  α) Με τον αλγόριθμο χρωματισμού γράφου ($k = 3$) για το γράφο παρεμβολής του σχήματος.



  β) Με τον αλγόριθμο της γραμμικής σάρωσης για τους χρόνους ζωής (A-F). Ο αριθμός των διαθέσιμων φυσικών καταχωρητών είναι $R = 3$.



Ενδεικτικά θέματα: Χρονοπρογραμματισμός κώδικα

1 Έστω η υποθετική αρχιτεκτονική RISC:

Διαμόρφωση	Συμπεριφορά	Κύκλοι μηχανής
ADD R1, R2, R3	R1 := R2 + R3	1
MUL R1, R2, R3	R1 := R2 * R3	2
LOAD R1, imm(R2)	R1 := MEM(R2 + imm)	3
STORE imm(R2), R1	MEM(R2 + imm) := R1	3

Να πραγματοποιηθεί ο χρονοπρογραμματισμός του παρακάτω τμήματος κώδικα ώστε ο συνολικός χρόνος εκτέλεσης σε αρχιτεκτονική με μία ALU και μία μονάδα (LOAD/STORE) να μειωθεί από τους 20 στους 13.

```
1-3:   LOAD  R1, 0(R30)
4-4:   ADD   R1, R1, R1
5-7:   LOAD  R2, 1(R30)
8-9:   MUL   R1, R1, R2
9-11:  LOAD  R3, 2(R30)
12-13: MUL   R1, R1, R3
13-15: LOAD  R2, 3(R30)
16-17: MUL   R1, R1, R2
18-20: STORE 4(R30), R1
```


Ενδεικτικά θέματα: Διάφορα

- Βελτιστοποιήσεις:
  α) Εφαρμόστε διαδοχικά δίπλωση σταθεράς, διάδοση σταθεράς, αλγεβρικές απλοποιήσεις και εξουδετέρωση κοινής υποεκφράσεως στο παρακάτω τμήμα κώδικα.

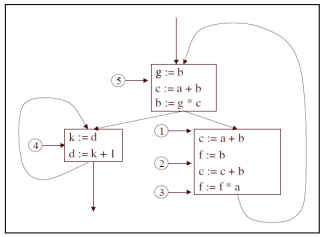
```
if (k == 0) {
  a = 11 + 1;
  b = a;
  c = (b + e) * 1024;
  d = e + b;
} else {
  x = 9 * a + c / 2;
}
```

  β) Να εφαρμοστεί πλακόστρωση βρόχων (loop tiling) για μέγεθος πλακιδίου ίσο με 16. Οι πίνακες a, b έχουν από n στοιχεία.

```
for (i = 0; i < n-1; i++) {
  b[i] += (a[i] + a[i+1])/2;
}
```


Θέματα εξάσκησης: Ανάλυση χρόνου ζωής

- Έστω το παρακάτω CFG. Να δοθούν τα σύνολα ζωντανών μεταβλητών στα σημεία 1 ως 5.



- Απάντηση

Live at 1 = {a, b, d}
Live at 2 = {a, b, c, d, f}
Live at 3 = {a, b, d}
Live at 4 = {k}
Live at 5 = {a, b, d, g}

- Μεταβλητές που μόνο διαβάζονται είναι ζωντανές πριν το σημείο εισόδου
- Ελέγξτε όλες τις διαδρομές

Θέματα εξάσκησης: CFG από δομημένο πηγαίο κώδικα

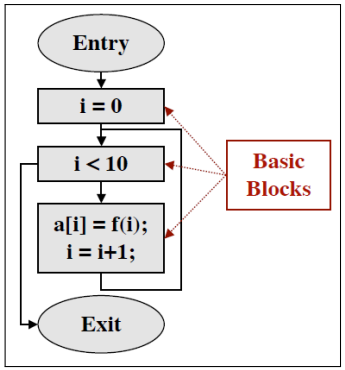
- Κώδικας ANSI C

```
extern int f(int);

int main(void)
{
  int i;
  int *a;

  for (i = 0; i < 10; i++)
  {
    a[i] = f(i);
  }
}
```

- Το CFG της συνάρτησης `main` με δηλώσεις C

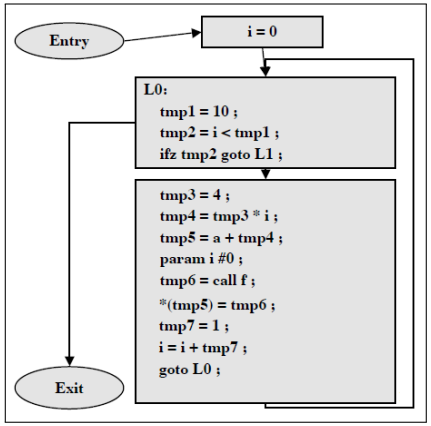


Θέματα εξάσκησης: CFG από κώδικα τριών διευθύνσεων

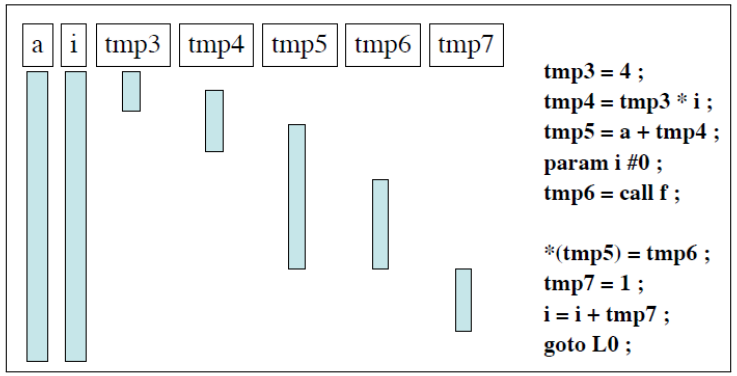
- Κώδικας TAC

```
main:
  i = 0;
L0:
  tmp1 = 10;
  tmp2 = i < tmp1;
  if (tmp2 == 0) then goto L1;
  tmp3 = 4;
  tmp4 = tmp3 * i;
  tmp5 = a + tmp4;
  param i, $0;
  tmp6 = call f;
  *tmp5 = tmp6;
  tmp7 = 1;
  i = i + tmp7;
  goto L0;
L1:
```

- Το CFG της συνάρτησης `main` με δηλώσεις C

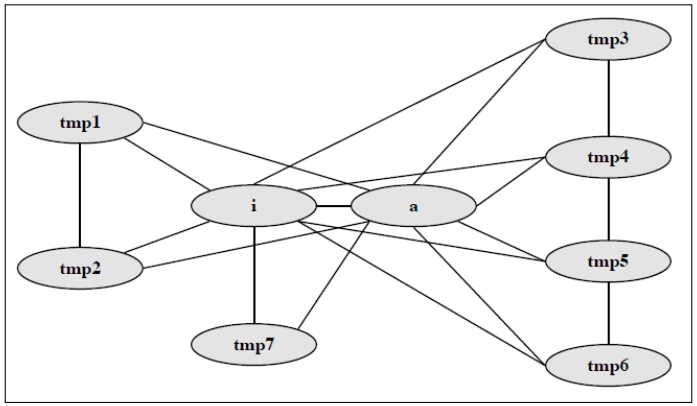


Θέματα εξάσκησης: Εύρεση διαστημάτων ζωής από τον κώδικα τριών διευθύνσεων



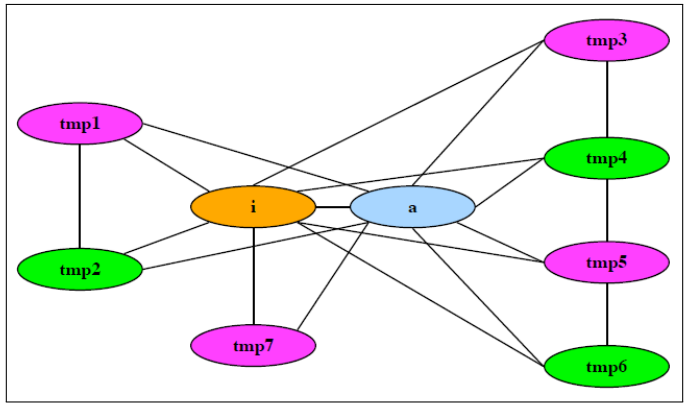
Θέματα εξάσκησης: Σχεδιασμός του γράφου παρεμβολής

Σαρώνουμε οριζόντια το προηγούμενο σχήμα και καταγράφουμε όλες τις διμελείς σχέσεις. Κάθε σχέση χρειάζεται να γραφεί μόνο μία φορά (όχι multi-graph)



Θέματα εξάσκησης: Αποτέλεσμα του χρωματισμού γράφου

Για αριθμό καταχωρητών $R = 4$



Η αρχιτεκτονική επεξεργαστή MIPS32

- Ο επεξεργαστής MIPS (Machine without Interlocked Pipeline Stages) αποτελεί έναν δημοφιλή RISC επεξεργαστή
- 32-bit δεδομένα και εντολές
- 3 διαφορετικές κωδικοποιήσεις εντολής
- Αρχιτεκτονικές MIPS-I (δεκαετίες ΄80, ΄90), MIPS32
- Αντίστοιχες υλοποιήσεις στο υλικό: MIPS R3000 και MIPS32-4k
- Υποστηρίζει τέσσερις συνεπεξεργαστές
- Συνήθως χρησιμοποιούνται οι
  - CP0 συνεπεξεργαστής συστήματος για την υποστήριξη Operating System
  - Coprocessor 1: FPU (Floating-Point Unit)
  - Coprocessor 2: VPU (Vector Processing Unit)
- Χρησιμοποιείται στις κονσόλες PlayStation, PSP, και PS2
- Σήμερα στην αγορά κυριαρχεί πλέον ο ARM (κινητή τηλεφωνία)

Ορόσημα στην εξέλιξη της αρχιτεκτονικής επεξεργαστή MIPS

Year	*Designer/model/ clock rate (MHz)*	*Instruction set*	*Cache (I+D)*	*Notes*
1987	MIPS R2000-16	MIPS I	External: 4 K+4 K to 32 K+32 K	External (R2010) FPU.
1990	IDT R3051-20		4 K+1 K	The first embedded MIPS CPU with on-chip cache and progenitor of a family of pin-compatible parts.
1991	MIPS R4000-100	MIPS III	8 K+8 K	Integrates FPU and L2 cache controller with pinout option. Full 64-bit CPU—but five years later, few MIPS CPUs were exploiting their 64-bit instruction set. Long pipeline and half-speed interface help achieve high clock rates.
1993	IDT/QED R4600-100		16 K+16 K	QED's brilliantly tuned redesign is much faster than R4000 or R4400 at the same clock rate—partly because it returned to the classic MIPS five-stage pipeline. Important to SGI's fast and affordable low-end Indy workstation and Cisco's routers.
1995	NEC/MIPS Vr4300-133		16 K+8 K	Low cost, low power but full-featured R4000 derivative. Initially aimed at Nintendo 64 games console, but embedded uses include HP's LJ4000 laser printers.
1996	MIPS R10000-200	MIPS IV	32 K+32 K	Bristling with microprocessor innovations, the R10000 is not at all simple. The main MIPS tradition it upholds is that of taking a principle to extremes. The result was hot, unfriendly, but with unmatched performance/MHz.
1998	QED RM7000		16 K+16 K+256 K L2	The first MIPS CPU with on-chip L2 cache, this powered generations of high-end laser printers and Internet routers.
2000	MIPS 4 K core family	MIPS32	16 K+16 K (typ)	The most successful MIPS core to date—synthesizable and frugal.
2001	Alchemy AU-1000		16 K+16 K	If you wanted 400 MHz for 500 mW, this was the only show in town. But it lost markets.
2001	Broadcom BCM1250	MIPS64	32 K+32 K+256 K L2	Dual-CPU design at 600 MHz+ (the L2 is shared).
2002	PMC-Sierra RM9000x2	MIPS64	16 K+16 K+256 K L2	Dual-CPU design at 1 GHz (the L2 is NOT shared; each CPU has its own 256 K). First MIPS CPU to reach 1 GHz.
2003	Intrinsity FastMath	MIPS32	16 K+16 K+1 M L2	Awesome 2-GHz CPU with vector DSP did not find a market.
2003	MIPS 24 K core	MIPS32 R2	At 500 MHz in synthesizable logic, a solidly successful core design.	
2005	MIPS 34 K core	MIPS32+MT ASE	32 K+32 K (typ)	MIPS multithreading pioneer.

Οι κωδικοποιήσεις των εντολών της αρχιτεκτονικής MIPS

- **All MIPS instructions are 32 bits long. 3 formats:**
  - **R-type**

31–26	25–21	20–16	15–11	10–6	5–0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

  - **I-type**

31–26	25–21	20–16	15–0
op	rs	rt	address/immediate
6 bits	5 bits	5 bits	16 bits

  - **J-type**

31–26	25–0
op	target address
6 bits	26 bits

- **The different fields are:**
  - **op: operation ("opcode") of the instruction**
  - **rs, rt, rd: the source and destination register specifiers**
  - **shamt: shift amount**
  - **funct: selects the variant of the operation in the "op" field**
  - **address / immediate: address offset or immediate value**
  - **target address: target address of jump instruction**

Το ρεπερτόριο εντολών της αρχιτεκτονικής MIPS (1)

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/ FUNCT (Hex)
Add	add	R	R[rd]=R[rs]+R[rt]	(1)	0/20
Add Immediate	addi	I	R[rt]=R[rs]+SignExtImm	(1)(2)	8
Add Imm. Unsigned	addiu	I	R[rt]=R[rs]+SignExtImm	(2)	9
Add Unsigned	addu	R	R[rd]=R[rs]+R[rt]	(2)	0/21
Subtract	sub	R	R[rd]=R[rs]-R[rt]	(1)	0/22
Subtract Unsigned	subu	R	R[rd]=R[rs]-R[rt]		0/23
And	and	R	R[rd]=R[rs]&R[rt]		0/24
And Immediate	andi	I	R[rt]=R[rs]&ZeroExtImm	(3)	c
Nor	nor	R	R[rd]=~(R[rs]|R[rt])		0/27
Or	or	R	R[rd]=R[rs]|R[rt]		0/25
Or Immediate	ori	I	R[rt]=R[rs]|ZeroExtImm	(3)	d
Xor	xor	R	R[rd]=R[rs]^R[rt]		0/26
Xor Immediate	xori	I	R[rt]=R[rs]^ZeroExtImm		e
Shift Left Logical	sll	R	R[rd]=R[rs]≪shamt		0/00
Shift Right Logical	srl	R	R[rd]=R[rs]≫shamt		0/02
Shift Right Arithmetic	sra	R	R[rd]=R[rs]≫>shamt		0/03
Shift Left Logical Var.	sllv	R	R[rd]=R[rs]≪R[rt]		0/04
Shift Right Logical Var.	srlv	R	R[rd]=R[rs]≫R[rt]		0/06
Shift Right Arithmetic Var.	srav	R	R[rd]=R[rs]≫>R[rt]		0/07
Set Less Than	slt	R	R[rd]=(R[rs]<R[rt])?1:0		0/2a
Set Less Than Imm.	slti	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)	a
Set Less Than Imm. Unsign.	sltiu	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)(6)	b
Set Less Than Unsigned	sltu	R	R[rd]=(R[rs]<R[rt])?1:0	(6)	0/2b
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/–/–/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/–/–/1b
Multiply	mult	R	{Hi,Lo}=R[rs]∗R[rt]		0/–/–/18
Multiply Unsigned	multu	R	{Hi,Lo}=R[rs]∗R[rt]	(6)	0/–/–/19

Το ρεπερτόριο εντολών της αρχιτεκτονικής MIPS (2)

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/ FUNCT (Hex)
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5
Branch Less Than	blt	P	if(R[rs]<R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than	bgt	P	if(R[rs]>R[rt]) PC=PC+4+BranchAddr		
Branch Less Than Or Equal	ble	P	if(R[rs]<=R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than Or Equal	bge	P	if(R[rs]>=R[rt]) PC=PC+4+BranchAddr		
Jump	j	J	PC=JumpAddr	(5)	2
Jump And Link	jal	J	R[31]=PC+4; PC=JumpAddr	(5)	2
Jump Register	jr	R	PC=R[rs]		0/08
Jump And Link Register	jalr	R	R[31]=PC+4; PC=R[rs]		0/09
Move	move	P	R[rd]=R[rs]		
Load Byte	lb	I	R[rt]={24'b0, M[R[rs]+ZeroExtImm](7:0)}	(3)	20
Load Byte Unsigned	lbu	I	R[rt]={24'b0, M[R[rs]+SignExtImm](7:0)}	(2)	24
Load Halfword	lh	I	R[rt]={16'b0, M[R[rs]+ZeroExtImm](15:0)}	(3)	25
Load Halfword Unsigned	lhu	I	R[rt]={16'b0, M[R[rs]+SignExtImm](15:0)}	(2)	25
Load Upper Imm.	lui	I	R[rt]={imm,16'b0}		f
Load Word	lw	I	R[rt]=M[R[rs]+SignExtImm]	(2)	23
Load Immediate	li	P	R[rd]=immediate		
Load Address	la	P	R[rd]=immediate		
Store Byte	sb	I	M[R[rs]+SignExtImm] (7:0)=R[rt](7:0)	(2)	28
Store Halfword	sh	I	M[R[rs]+SignExtImm] (15:0)=R[rt](15:0)	(2)	29
Store Word	sw	I	M[R[rs]+SignExtImm]=R[rt]	(2)	2b

Οι καταχωρητές γενικού σκοπού του MIPS

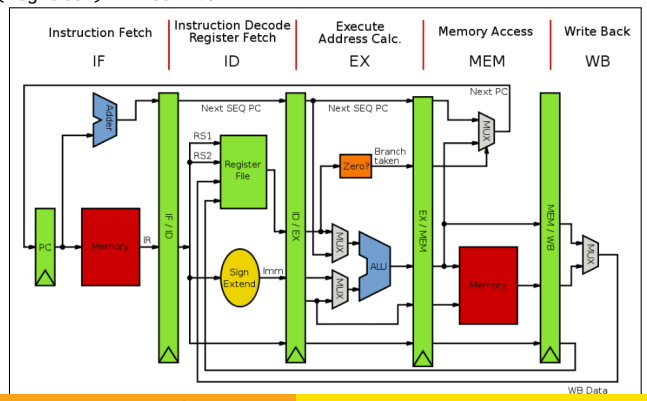
Ο MIPS διαθέτει 32 καταχωρητές γενικού σκοπού

Register Name	Software Name (from regdef.h)	Use and Linkage
$0		Always has the value 0.
$at		Reserved for the assembler.
$2..$3	v0-v1	Used for expression evaluations and to hold the integer type function results. Also used to pass the static link when calling nested procedures.
$4..$7	a0-a3	Used to pass the first 4 words of integer type actual arguments, their values are not preserved across procedure calls.
$8..$15	t0-t7	Temporary registers used for expression evaluations; their values aren't preserved across procedure calls.
$16..$23	s0-s7	Saved registers. Their values must be preserved across procedure calls.
$24..$25	t8-t9	Temporary registers used for expression evaluations; their values aren't preserved across procedure calls.
$26..27 or $kt0..$kt1	k0-k1	Reserved for the operating system kernel.
$28 or $gp	gp	Contains the global pointer.
$29 or $sp	sp	Contains the stack pointer.
$30 or $fp	fp	Contains the frame pointer (if needed); otherwise a saved register (like s0-s7).
$31	ra	Contains the return address and is used for expression evaluation.

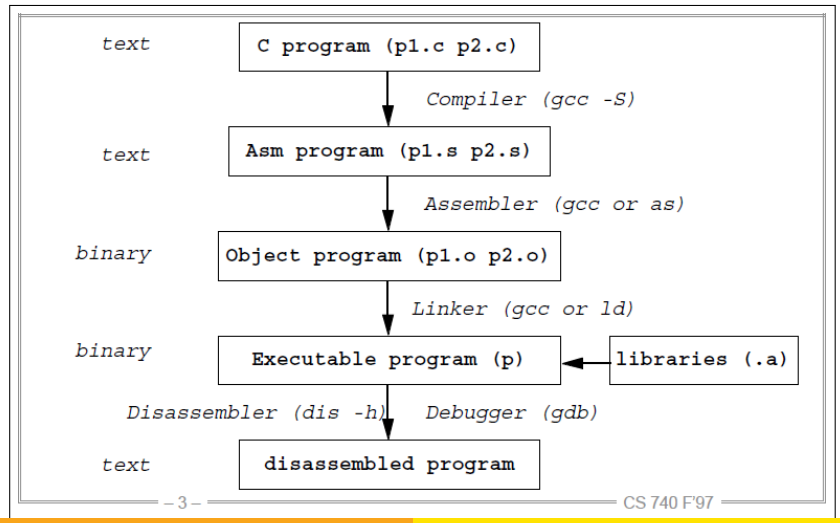
Οργάνωση διοχέτευσης ενός επεξεργαστή MIPS (MIPS-R3000)

Ο MIPS-R3000 διαθέτει 5 βαθμίδες διοχέτευσης:

- IF: Instruction Fetch
- ID: Instruction Decode
- EX: Execute
- MEM: Memory Access
- WB: (Register) Write-Back

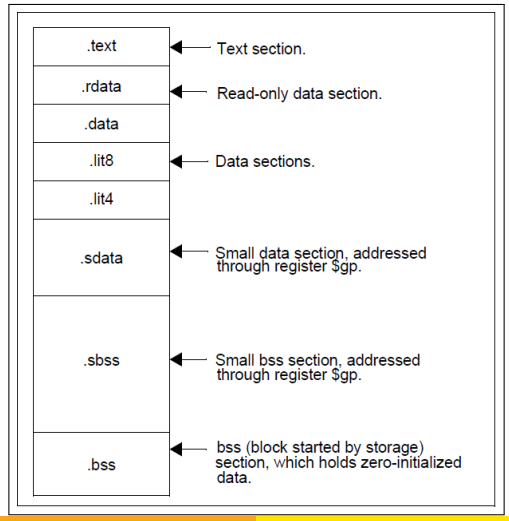


Η διαδικασία της μεταγλώττισης πηγαίου κώδικα για τον MIPS



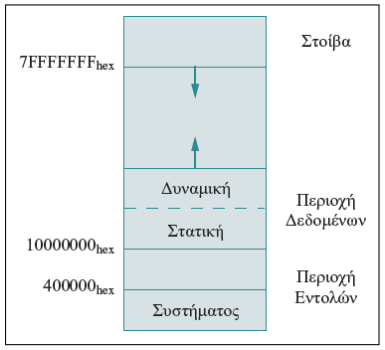
Η οργάνωση των αντικείμενων αρχείων (object files) τύπου ELF για τον MIPS

ELF: Executable Linking Format

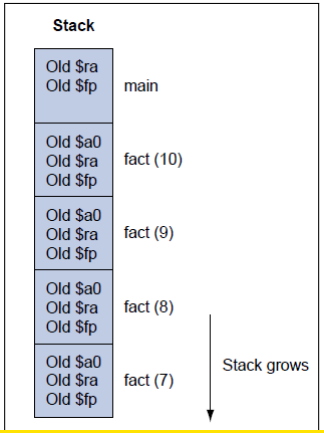


Οργάνωση και διαχείριση της μνήμης δεδομένων

Χάρτης μνήμης για τον MIPS



Διαχείριση στοίβας για αναδρομικές κλήσεις (παραγοντικό)



Γενικοί κανόνες που ακολουθούνται στον γεννήτορα τελικού κώδικα

- Υπολογισμός του συνολικού μεγέθους των τοπικών και προσωρινών μεταβλητών ενός τοπικού πίνακα συμβόλων
- Υπολογισμός του χώρου που καταλαμβάνουν οι παράμετροι των συναρτήσεων και η σχετική τους θέση
- Ο `$sp` δείχνει στην τελευταία θέση στη στοίβα και ο `$fp` στην πρώτη τοπική μεταβλητή στη στοίβα
- Όλες οι τοπικές μεταβλητές και οι παράμετροι των συναρτήσεων τοποθετούνται στη στοίβα του προγράμματος: `<relative parameter/variable order * 4>($fp)`
- Για μεταγλώττιση χωρίς βελτιστοποιήσεις χρησιμοποιήστε το μοντέλο φόρτωσης { υπολογισμού { αποθήκευσης
- Κάθε γενική (global) μεταβλητή αποτελεί σύμβολο στη μνήμη και δηλώνεται με ψευδοεντολή

```
.data
<name>: <type> <size>
```

- Για κάθε string δηλώνονται τιμές αρχικοποίησης με ψευδοεντολές `.word`

Δημιουργία συμβολικού κώδικα για συναρτήσεις

- Υπολογισμός συνολικού μεγέθους για μεταβλητές, καταχωρητές, και το πλαίσιο (frame) της συνάρτησης (τοπικές μεταβλητές, αποθήκευση των `$s0 to $s7`, `$fp`, `$ra`
- Ετικέτα συνάρτησης

```
.text
L_<function name>:
```

- Δημιουργία του πλαισίου της συνάρτησης: `subu $sp, <frame size>`
- Αποθήκευση καταχωρητών: `sw <register name>, <relative position>($sp)`
- Νέα τιμή του `$fp`: `addu $fp, $sp, <frame size>`
- Κώδικας για το σώμα της συνάρτησης
- Κώδικας επιλόγου
  - Για συνάρτηση με επιστρεφόμενη τιμή: `lw $v0, <position in memory>`
  - Επαναφορά των τιμών καταχωρητών: `sw <register name>, <relative position>($sp)`
  - Μείωση της τιμής του `$sp`: `subu $sp, $sp, <frame size>`
  - Αλλαγή του σημείου εκτέλεσης του προγράμματος: `jal $ra`

Γέννηση κώδικα για MIPS: Σύγκριση συμβολοσειρών

ANSI C

```
int strcmp (char *str1, char *str2)
{
  char c1, c2;

  do {
    c1 = *str1++;
    c2 = *str2++;
  } while (c1 != 0 && c2 != 0 && c1 == c2);

  /* clever: 0, +ve or -ve as required */
  return c1 - c2;
}
```

Κώδικας assembly για τον MIPS

```
strcmp:
1:
    lbu   t0, 0(a0)
    addu  a0, a0, 1
    lbu   t1, 0(a1)
    addu  a1, a1, 1
    beq   t0, zero, .t01
# end of first string?
    beq   t1, zero, .t01
# end of second string?
    beq   t0, t1, 1b
.t01:
    subu  v0, t0, t1
    j     ra
```


Γέννηση κώδικα για MIPS: Βρόχος for (πολλαπλασιασμός)

ANSI C

```
int test_for(int a, int b)
{
  int t = 0, i;
  for (i = 0; i < b; i++)
  {
    t += a;
  }
  return t;
}
```

Assembly

```
.text
.align 2
test_for:
.frame sp, 0, r31
move r2, r0
blez r5, .L7
.L5:
addiu r5, r5, -1
addu r2, r2, r4
bne     r5, r0, .L5
.L7:
j     r31
```


Γέννηση κώδικα για MIPS: Αναδρομικές κλήσεις συναρτήσεων στον υπολογισμό παραγοντικού

ANSI C

```
int fact(int n) {
  if (n > 1)
    return (n * fact(n-1));
  else
    return 1;
}
```

Κώδικας assembly για τον MIPS

```
fact:
    addi   $sp, $sp, -8     # make space on stack for 2 items
    sw     $ra, 4($sp)      # push return address (from ra)
    sw     $a0, 0($sp)      # push n (from a0)
    sgti   $t0, $a0, 1      # if (n > 1)
    bne    $t0, $zero, rec  # goto rec;
                            # else
    addi   $v0, $zero, 1    # v0 = 1; (value to return)
    addi   $sp, $sp, 8      # hand back stack space (no pop)
    jr     $ra              # return to caller
rec:
    addi   $a0, $a0, -1     # a0 = n - 1;
rcall:
    jal    fact             # v0 = fact(n - 1);
    lw     $a0, 0($sp)      # pop n (into a0)
    lw     $ra, 4($sp)      # pop return address (into ra)
    addi   $sp, $sp, 8      # hand back stack space
    mul    $v0, $a0, $v0    # v0 = n * fact(n - 1)); (ret val)
    jr     $ra              # return to caller $
```

Γέννηση κώδικα για MIPS: Παραγοντοποίηση σε γινόμενο πρώτων

ANSI C

```c
void pfactor(unsigned int x,
  unsigned int *outp)
{
  unsigned int i, n;
  n = x;
  i = 2;
  while (i <= n) {
   while ((n % i) == 0) {
      n = n / i;
      *outp = i;
    }
    i = i + 1;
  }
}
```

```
.text
.align 2
pfactor:
.frame  sp, 0, R31
sltu    R2, R4, 2
li      R3, 2
bne     R2, R0, .L9
.L20:
divu    R0, R4, R3
teq     R3, R0, 7
mfhi    R2
bne     R2, R0, .L11
.L14:
divu    R0, R4, R3
teq     R3, R0, 7
mflo    R4
divu    R0, R4, R3
teq     R3, R0, 7
mfhi    R6
beq     R6, R0, .L14
sw      R3, 0(R5)
.L11:
addiu   R3, R3, 1
sltu    R2, R4, R3
beq     R2, R0, .L20
.L9:
j       R31
```