

Μεταγλωττιστές II

Γέννηση τελικού κώδικα για RISC επεξεργαστές

Νικόλαος Καββαδίας
nkavn@uop.gr

12 Ιανουαρίου 2011

Σκιαγράφηση της διάλεξης

- Η αρχιτεκτονική επεξεργαστή MIPS
- Γέννηση τελικού κώδικα για τον επεξεργαστή MIPS
- Παραδείγματα μεταγλώττισης κώδικα ANSI C για την αρχιτεκτονική MIPS32
- Ο διερμηνευτικός προσομοιωτής `spim`

Η αρχιτεκτονική επεξεργαστή MIPS32

[Patterson, 2009, Sweetman, 2005]

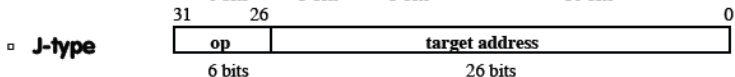
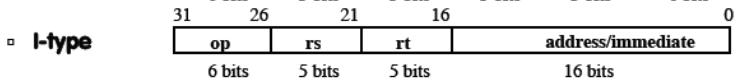
- Ο επεξεργαστής MIPS (Machine without Interlocked Pipeline Stages) αποτελεί έναν δημοφιλή επεξεργαστή RISC
- 32-bit δεδομένα και εντολές
- 3 διαφορετικές κωδικοποιήσεις εντολής
- Αρχιτεκτονικές MIPS-I (δεκαετίες '80, '90), MIPS32
- Αντίστοιχες υλοποιήσεις στο υλικό: MIPS R3000 και MIPS32-4k
- Υποστηρίζει τέσσερις συνεπεξεργαστές
- Συνήθως χρησιμοποιούνται οι
 - CP0 συνεπεξεργαστής συστήματος για την υποστήριξη λειτουργικού συστήματος
 - 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:**



- **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	$R[rd]=\sim(R[rs])\&R[rt]$	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	$R[rd]=R[rs]\llshamt$	0/00
Shift Right Logical	srl	R	$R[rd]=R[rs]\ggshamt$	0/02
Shift Right Arithmetic	sra	R	$R[rd]=R[rs]\ggshamt$	0/03
Shift Left Logical Var.	sllv	R	$R[rd]=R[rs]\llR[rt]$	0/04
Shift Right Logical Var.	srlv	R	$R[rd]=R[rs]\ggR[rt]$	0/06
Shift Right Arithmetic Var.	srav	R	$R[rd]=R[rs]\ggR[rt]$	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.

Κανόνες επιλογής κώδικα για τον MIPS με ταύτιση μοτίβων δένδρου (1)

TEMP	t_n
LABEL	label:
CONST(0)	zero
CONST(CONST_16)	addi Rd, zero, I_16
CONST(*)	li Rd, *
JUMP(NAME)	b label
JUMP(*)	jr Rs
NAME	la Rd, label
MOVE(MEM(+(*, CONST_16)), *)	sw Rs, I_16(Rd)
MOVE(MEM(+ (CONST_16, *)), *)	sw Rs, 0(Rd)
MOVE(MEM(*), *)	move Rd, Rs
BINOP(PLUS, *, *)	add Rd, Rs1, Rs2
BINOP(PLUS, *, CONST_16)	addi Rd, Rs, I_16
BINOP(PLUS, CONST_16, *)	
BINOP(MINUS, *, *)	sub Rd, Rs1, Rs2
BINOP(MINUS, *, CONST_16)	addi Rd, Rs, -I_16
BINOP(MUL, *, *)	mulo Rd, Rs1, Rs2
BINOP(MUL, *, CONST_2^k)	sll Rd, Rs, I_k
BINOP(MUL, CONST_2^k, *)	

Κανόνες επιλογής κώδικα για τον MIPS με ταύτιση μοτίβων δένδρου (2)

<code>BINOP(DIV,*,*)</code>	<code>div</code>	<code>Rd, Rs1, Rs2</code>
<code>BINOP(DIV,*,CONST_2^k)</code>	<code>sra</code>	<code>Rd, Rs, I_k</code>

<code>BINOP(AND,*,*)</code>	<code>and</code>	<code>Rd, Rs1, Rs2</code>
<code>BINOP(AND,*,CONST_16),</code>	<code>andi</code>	<code>Rd, Rs, I_16</code>
<code>BINOP(AND,CONST_16,*)</code>		
same for OR, BITXOR		

<code>BINOP(LSHIFT,*,*)</code>	<code>sllv</code>	<code>Rd, Rs1, Rs2</code>
<code>BINOP(LSHIFT,*,CONST_16)</code>	<code>sll</code>	<code>Rd, Rs, I_16</code>

<code>BINOP(RSHIFT,*,*)</code>	<code>srlv</code>	<code>Rd, Rs1, Rs2</code>
<code>BINOP(RSHIFT,*,CONST_16)</code>	<code>srl</code>	<code>Rd, Rs, I_16</code>

<code>BINOP(ARSHIFT,*,*)</code>	<code>srav</code>	<code>Rd, Rs1, Rs2</code>
<code>BINOP(ARSHIFT,*,CONST_16)</code>	<code>sra</code>	<code>Rd, Rs, I_16</code>

<code>MEM(+ (CONST_16,*)) ,</code>	<code>lw</code>	<code>Rd, I_16(Rs)</code>
<code>MEM(+(*,CONST_16))</code>		
<code>MEM(*)</code>	<code>lw</code>	<code>Rd, 0(Rs)</code>

<code>CJUMP(op,*,*,label,*)</code>	<code>bx</code>	<code>Rs1, Rs2, label</code>
where: <code>bx</code> = { <code>beq</code> , <code>bne</code> , <code>blt</code> , <code>bgt</code> , <code>ble</code> , <code>bge</code> }		

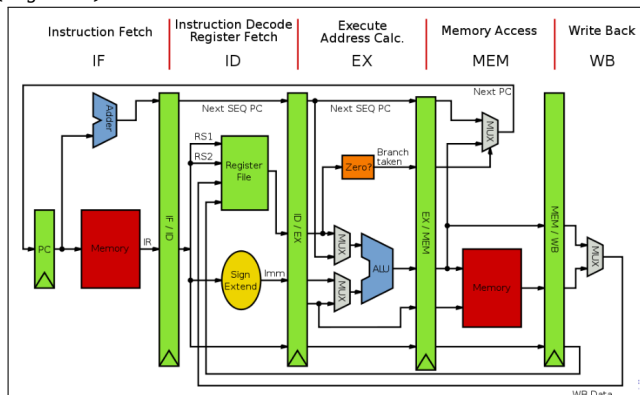
<code>JUMP(NAME,[*])</code>	<code>b</code>	<code>label</code>
<code>JUMP(*,[*])</code>	<code>jr</code>	<code>Rs</code>

<code>CALL(NAME,[*])</code>	<code>jal</code>	<code>label</code>

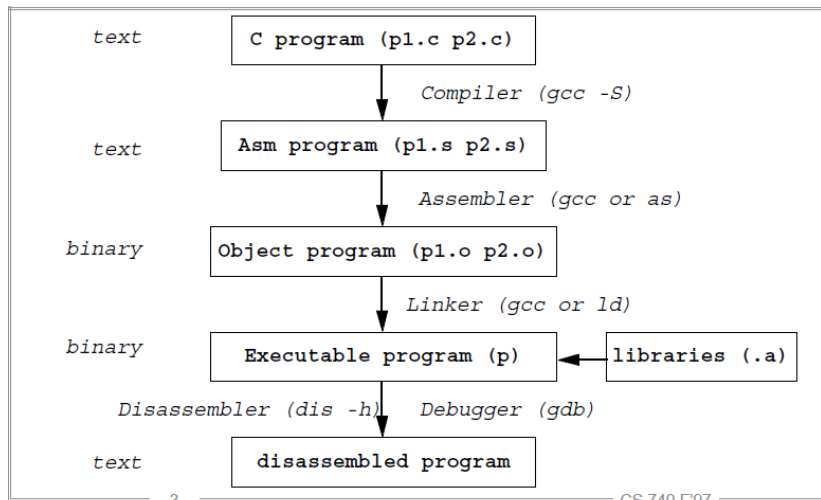
Οργάνωση διοχέτευσης ενός επεξεργαστή MIPS (MIPS-R3000)

Ο MIPS-R3000 διαθέτει 5 βαθμίδες διοχέτευσης:

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

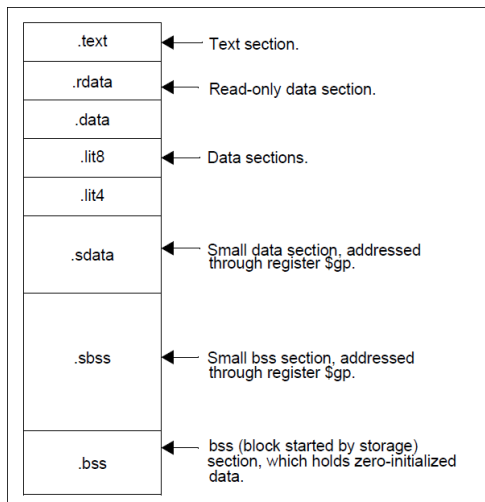


Η διαδικασία της μεταγλώττισης πηγαίου κώδικα για τον MIPS



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

ELF: Executable Linking Format



Χωροθέτηση κώδικα προγράμματος στη μνήμη: Παράδειγμα

ANSI C

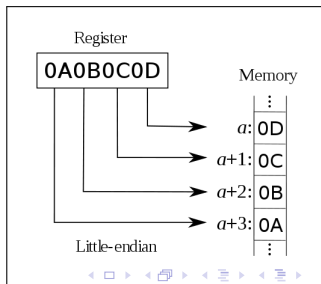
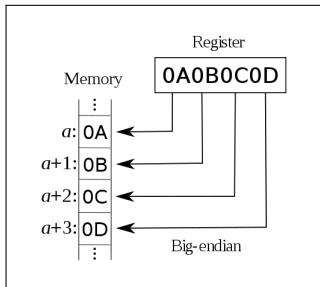
```
int x;  
char y, z;  
  
foo()  
{  
    ...  
}  
  
int a[10];  
  
bar()  
{  
    ...  
}
```

Assembly

```
.global x  
x: .space 4  
.global y  
y : .space 1  
.global z  
z : .space 1  
.align 2  
.global foo  
foo:  
    (code for foo)  
  
.global a  
a: .space 40  
  
.global bar  
bar:  
    (code for bar)
```

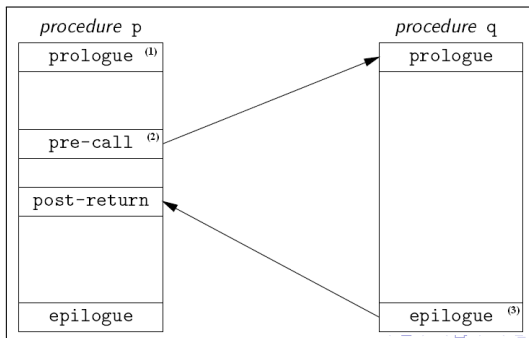
Οργάνωση των δεδομένων στους πόρους αποθήκευσης

- Δύο τρόποι για την οργάνωση των δεδομένων κατά την αποθήκευση τους στη μνήμη
 - *endianness*: Η σχετική διάταξη των byte, κατά αύξουσες ή μειούμενες διευθύνσεις
 - Χιουμοριστικός όρος από τα 'Ταξίδια του Γκιούλιβερ' αναφερόμενος στη διχογνωμία για το ποιος είναι ο κατάλληλος τρόπος οργάνωσης
- Big-endian: MIPS-I, PowerPC, SPARC
 - Little-endian: Intel 8086, Pentium, Alpha



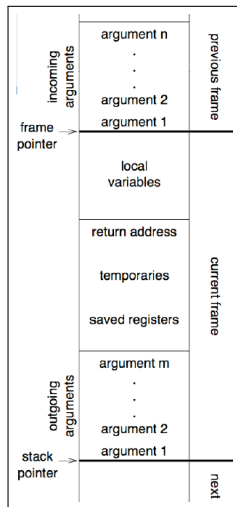
Γενικό μοντέλο κλήσης υπορουτινών

- Έστω δύο υπορουτίνες, p , q , με την p να καλεί την q
- κατά την είσοδο στην p εκτελείται κώδικας επαναφοράς του περιβάλλοντός της
- όταν καλείται η q , διατηρείται το περιβάλλον της p
- κατά την έξοδο από την q , επαναφέρεται το περιβάλλον της p



Κλήση υπορουτινών με σύνδεση

- 1 ο καλών ωθεί χώρο για την τιμή επιστροφής
- 2 ο καλών ωθεί το δείκτη στοιβας
- 3 ο καλών ωθεί χώρο για: διεύθυνση επιστροφής, τη διεύθυνση στη στατική ακολουθία προσωρινών μεταβλητών και τους σωζόμενους καταχωρητές
- 4 ο καλών ωθεί τα παραπάνω στη στοιβα
- 5 ο καλών θέτει τη διεύθυνση επιστροφής και τη στατική ακολουθία του καλούμενου και πραγματοποιεί την κλήση
- 6 ο καλούμενος αποθ. καταχωρητές σε ειδική περιοχή
- 7 ο καλούμενος αντιγράφει by-value πίνακες/εγγραφές χρησιμοποιώντας διευθύνσεις που του έγιναν γνωστές
- 8 ο καλούμενος εκχωρεί δυναμική μνήμη για πίνακες κατά απαίτηση
- 9 κατά την επιστροφή, ο καλούμενος επαναφέρει τους αποθηκευμένους καταχωρητές
- 10 ο καλούμενος πραγματοποιεί άλμα στη διεύθυνση επιστροφής

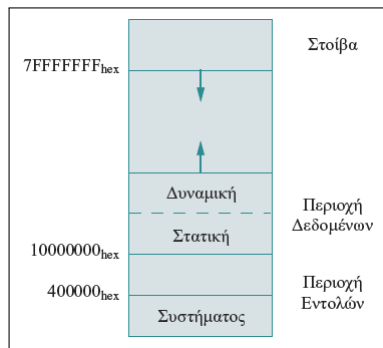


Συμβάσεις κλήσεως C ρουτινών (C calling convention)

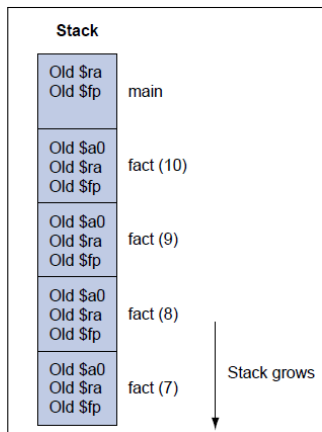
- Stack Management
 - The stack grows down
 - Subtract from `$sp` to allocate local storage space
 - Restore `$sp` by adding the same amount at function exit
 - The stack must be 8-byte aligned
 - Modify `$sp` only in multiples of eight
- Function Parameters
 - Every parameter smaller than 32 bits is promoted to 32 bits
 - First four parameters are passed in registers `$a0–$a3`
 - 64-bit parameters are passed in register pairs
 - Every subsequent parameter is passed through the stack
 - First 16 bytes on the stack are not used
 - Assuming `$sp` was not modified at function entry:
 - The 1st stack parameter is located at `16($sp)`.
 - The 2nd stack parameter is located at `20($sp)`, etc.
 - 64-bit parameters are 8-byte aligned
- Return Values
 - 32-bit and smaller values are returned in register `$v0`
 - 64-bit values are returned in registers `$v0` and `$v1`

Οργάνωση και διαχείριση της μνήμης δεδομένων

Χάρτης μνήμης για τον MIPS



Διαχείριση στοίβας για αναδρομικές κλήσεις (παραγοντικό)



Γενικοί κανόνες που ακολουθούνται στο 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`

Κώδικας συμβολομεταφραστή για τον MIPS: Εύρεση μεγαλύτερου στοιχείου πίνακα

```
                                # a0: address of element mem[0]
                                # a1: value to match
prog:
  add  $t0, $zero, $zero # t0 = 0
  add  $v0, $zero, $zero # v0 = 0
  add  $v1, $zero, $zero # v1 = 0
loop:
  sltu $t2, $t0, $a1      # t2 = (t0 < a1)
  beq  $t2, $zero, fin    # if (!t2) goto fin
  lw   $t1, 0($a0)        # t1 = mem[a0]
  sltu $t2, $t1, $v0      # t2 = (t1 < v0) UNSIGNED!
  bne  $t2, $zero, skip   # if (t2) goto skip
  add  $v0, $t1, $zero    # v0 = t1
  add  $v1, $t0, $zero    # v1 = t0
skip:
  addi $t0, $t0, 1        # t0 = t0 + 1
  addi $a0, $a0, 4        # a0 = a0 + 4
  j    loop               # goto loop
fin:
                                # $finish
```

Το πρόγραμμα αυτό βρίσκει το μεγαλύτερο στοιχείο σε έναν πίνακα απρόσημων 32-bit ακεραίων και επιστρέφει το στοιχείο αυτό στον καταχωρητή \$v0 και τη θέση στον πίνακα στον \$v1

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 (από mips-sde-gcc, εφαρμόζει loop reversal)

```
.text
.align 2
test_for:
    .frame $sp, 0, $ra
    move   r2, $zero      # r2 = $v0
    blez  r5, .L7
.L5:
    addiu  r5, r5, -1
    addu   r2, r2, r4      # r4 = $a0
    bne   r5, $zero, .L5
.L7:
    jr     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

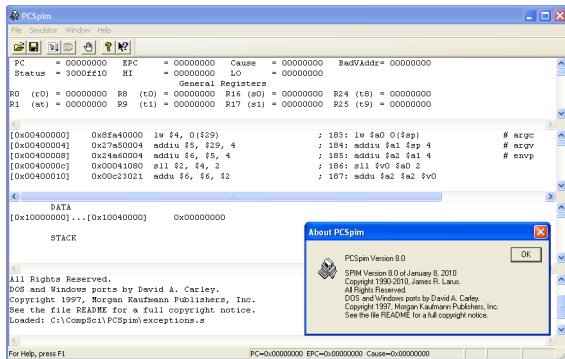
```
void pfactor(unsigned int x,
             unsigned int *outp)
{
    unsigned int i, n;
    n = x;
    i = 2;
    while (i <= n) {
        while ((n % i) == 0) {
            n = n / i;
            *outp = i;
        }
        i = i + 1;
    }
}
```

Assembly

```
.text
.align 2
pfactor:
    .frame R29, 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
```

Το γραφικό περιβάλλον του PCSpin [SPIM]

- **spim**: πολύ γνωστός διερμηνευτικός προσομοιωτής για τις αρχιτεκτονικές MIPS-I, MIPS32. Αποτελεί έργο του James Larus και βρίσκεται σε διαρκή ανάπτυξη από το 1990
- Άλλοι προσομοιωτές: MARS [MARS, Vollmar, 2006], EduMIPS64, WinDLX



Ψευδοεντολές (pseudoinstructions) στον spim

- Πρόκειται για ντιρεκτίβες (οδηγίες) που απευθύνει ο προγραμματιστής προς το συμβολομεταφραστή του spim

<code>.data [addr]*</code>	Subsequent items are stored in the data segment
<code>.kdata [addr]*</code>	Subsequent items are stored in the kernel data segment
<code>.ktext [addr]*</code>	Subsequent items are stored in the kernel text segment
<code>.text [addr]*</code>	Subsequent items are stored in the text * starting at [addr] if specified
<code>.ascii str</code>	Store string <i>str</i> in memory, but do not null-terminate it
<code>.asciiz str</code>	Store string <i>str</i> in memory and null-terminate it
<code>.byte b₁, ..., b_n</code>	Store the <i>n</i> values in successive bytes of memory
<code>.double d₁, ..., d_n</code>	Store the <i>n</i> floating-point double precision numbers in successive memory locations
<code>.float f₁, ..., f₁</code>	Store the <i>n</i> floating-point single precision numbers in successive memory locations
<code>.half h₁, ..., h_n</code>	Store the <i>n</i> 16-bit quantities in successive memory halfwords
<code>.word w₁, ..., w_n</code>	Store the <i>n</i> 32-bit quantities in successive memory words
<code>.space n</code>	Allocate <i>n</i> bytes of space in the current segment
<code>.extern symsize</code>	Declare that the datum stored at <i>sym</i> is <i>size</i> bytes large and is a global label
<code>.globl sym</code>	Declare that label <i>sym</i> is global and can be referenced from other files
<code>.align n</code>	Align the next datum on a 2 ⁿ byte boundary, until the next <code>.data</code> or <code>.kdata</code> directive
<code>.set at</code>	Tells SPIM to complain if subsequent instructions use <code>\$at</code>
<code>.set noat</code>	prevents SPIM from complaining if subsequent instructions use <code>\$at</code>

Κλήσεις συστήματος (system calls, syscalls) στον spim

- Επικοινωνία του προγράμματος του χρήστη με το περιβάλλον εκτέλεσης (λειτουργικό σύστημα)
 - είσοδος ή έξοδος ακέραιου αριθμού, αριθμού κινητής υποδιαστολής, συμβολοσειράς
 - άμεση έξοδος (exit) από το πρόγραμμα
 - δυναμική εκχώρηση μνήμης με τη βοήθεια της sbrk

SERVICE	\$v0	ARGS	RESULT
print_int	1	integer \$a0	
print_float	2	float \$f12	
print_double	3	double \$f12/\$f13	
print_string	4	string \$a0	
read_int	5		integer (in \$v0)
read_float	6		float (in \$f0)
read_double	7		double (in \$f0)
read_string	8	buf \$a0, buflen \$a1	
sbrk	9	amount \$a	address (in \$v0)
exit	10		

Παραδείγματα χρήσης των κλήσεων συστήματος

Εκτύπωση του ακεραίου που βρίσκεται στον καταχωρητή \$t2

```
li $v0, 1 # load syscall code in v0
move $a0, $t2 # move integer to be printed into a0
syscall # call operating system to perform operation
```

Ανάγνωση ακεραίου (από την κονσόλα) και αποθήκευσή του στη μνήμη

```
li $v0, 5 # load syscall code in v0
syscall # call operating system to perform operation
sw $v0, int_value # value read from keyboard returned in v0
# then stored to desired location
```

Εκτύπωση συμβολοσειράς

```
.data
str1 .asciiz "Print this.\n"
.text
main: li $v0, 4 # load appropriate system call code into register v0
      la $a0, str1 # load address of string in a0
      syscall # call operating system to perform print
```

Έξοδος από το πρόγραμμα

```
li $v0, 10 # system call code for exit = 10
syscall # call operating system
```

Αναφορές του μαθήματος I



David A. Patterson and John L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 4th Edition, Morgan-Kaufmann, 2009.



Dominic Sweetman, *See MIPS Run*, Second Edition, Morgan-Kaufmann, 2005.



Kenneth Vollmar and Pete Sanderson, “MARS: An Education-Oriented MIPS Assembly Language Simulator,” *ACM SIGCSE Bulletin*, vol. 38, no. 1, pp. 239-243, March 2006.



MARS (MIPS Assembler and Runtime Simulator): An IDE for MIPS Assembly Language Programming. [Online]. Available: <http://courses.missouristate.edu/KenVollmar/MARS/>



SPIM MIPS Simulator. [Online]. Available: <http://www.cs.wisc.edu/~larus/spim.html>