



Ingegneria degli Algoritmi (A.A. 2013-2014)

Corsi di Laurea in Ingegneria Informatica e Automatica, Ingegneria dei Sistemi Informatici, e Laurea Magistrale in Ingegneria Informatica

Sapienza Università di Roma

Appello del 17/06/2014 – Compito A – Durata 45'

Domanda 1 (3 punti)

Si consideri un programma formato da due moduli `sum.c` e `main.c`, una header `sum.h` inclusa da `sum.c` e da `main.c`, e una header `config.h` inclusa da `sum.c`. Scrivere un `Makefile` per compilare il programma con `gcc` in modo che ciascun modulo venga compilato separatamente e il tutto venga poi linkato generando un file eseguibile `test`.

Domanda 2 (2 punti)

Per ciascuna delle seguenti espressioni si dica se è valida e, in caso affermativo, si stabilisca se denota un oggetto (Lvalue) o un valore (Rvalue) e se ne determini il tipo. Si assuma che siano definite `int x`, `int** p` ed `int (*m)[2]`:

- 1) `*(&x+10)+7` 2) `*(p**p)` 3) `&m**m` 4) `(*m)[x]`

Domanda 3 (2 punti)

Si descrivano le seguenti tecniche di ottimizzazione, fornendo esempio per ciascuna almeno un esempio di applicazione: *strength reduction*, *constant folding*, *common subexpression elimination*, *loop fusion*.

Domanda 4 (3 punti)

Tradurre in C la seguente funzione x86-64, ricordando che nel System V AMD64 ABI il primo parametro è nel registro `rdi` e il valore di ritorno nel registro `rax`:

```
f:  cpl  $0, %edi
   je   L2
   xorl %eax, %eax      # equivalente a movl $0, %eax
   movl $1, %ecx
L1: addl %ecx, %eax
   incl %ecx
   cpl  %edi, %ecx
   jbe  L1
   ret
L2: xorl %eax, %eax
   ret
```

Fare riferimento alla tabella in calce al compito. Che cosa calcola la funzione?

prefix	description	example	C analog
add	add source to destination	addl \$5,%ecx	ecx += 5
call	procedure call	call _foo	foo()
cltq	sign-extend eax to rax	–	–
dec	decrement destination	decq %rcx	rcx--
imul	multiply destination with source	imull %esi,%eax	eax *= esi
inc	increment destination	incl %ecx	ecx++
ja	jump if above <i>(unsigned comparison)</i>	cmpl %eax,%ebx ja L2	if ((unsigned)eax < (unsigned)ebx) goto L2
jae	jump if above or equal <i>(unsigned comparison)</i>	cmpl %eax,%ebx jae L2	if ((unsigned)eax <= (unsigned)ebx) goto L2
jb	jump if below <i>(unsigned comparison)</i>	cmpl %eax,%ebx jb L2	if ((unsigned)eax > (unsigned)ebx) goto L2
jbe	jump if below or equal <i>(unsigned comparison)</i>	cmpl %eax,%ebx jbe L2	if ((unsigned)eax >= (unsigned)ebx) goto L2
je	jump if equal	cmpq %rax,%rbx je L2	if (rax == rbx) goto L2
jg	jump if greater <i>(signed comparison)</i>	cmpq %rax,%rbx jg L2	if (rax < rbx) goto L2
jge	jump if greater or equal <i>(signed comparison)</i>	cmpq %rax,%rbx jge L2	if (rax <= rbx) goto L2
jl	jump if less <i>(signed comparison)</i>	cmpq %rax,%rbx jl L2	if (rax > rbx) goto L2
jle	jump if less or equal <i>(signed comparison)</i>	cmpq %rax,%rbx jle L2	if (rax <= rbx) goto L2
jmp	unconditional jump	jmp L2	goto L2
jne	jump if not equal	cmpq %rax,%rbx jne L2	if (rax != rbx) goto L2
jnz	identical to jne	–	–
jz	identical to je	–	–
lea	copy address to destination (load effective address)	leaq -12(%rax),%rcx	rcx=rax-12
leave	pop the current stack frame, and restore the caller's frame	–	–
mov	copy data from source to destination	movq \$7,(%rax)	*(long*)rax=7
movabs	copy 64-bit immediate to destination register	movabsq \$-7,%rax	rax=-7
movsl	copy sign-extended word to destination register	movslq %bx,%rax	rax=bx
movzlw	copy zero-extended word to destination register	movzlw %bx,%eax	eax=(unsigned)bx
movsb	copy sign-extended byte to destination register	movsbq %bl,%rax	rax=bl
movzb	copy zero-extended byte to destination register	movzbq %bl,%rax	rax=(unsigned)bl
pop	pop value from stack and write it to destination	popq %rbx	–
push	push value on stack	pushq %rbx	–
ret	return from procedure call	ret	return
sub	subtract source from destination	subl \$5,%ecx	ecx -= 5

Tabella 1: Istruzioni x86-64 più comunemente utilizzate.