Università di Trieste
Facoltà di Scienze MFN
Corso di laurea in Informatica
a.a. 2010-2011

Corso "Architetture elaboratori"
Complementi di Assembler

Prima parte: uso delle reti

Tratto dal programma di “Laboratorio di informatica multimediale” a.a. 2006-2007

Programma-manifesto del corso

Il significato dell' aggettivo "multimediale" ha subito un notevole cambiamento dal 1980 ad oggi.

Nato inizialmente nell' industria culturale per significare l' uso sinergico di diversi mezzi di comunicazione di massa, ognuno dei quali aveva caratteristiche di produzione, diffusione e fruizione diverse - la radio, la TV, la stampa, l' affissione, la pubblicazione di audio e videocassette – si è via via trasformato sino a significare l' uso contemporaneo di segnali destinati ad aree percettive diverse del nostro cervello: il testo scritto, il testo parlato, la musica, la foto, il grafico, il filmato. (Non sono ancora stati pienamente inclusi fra i "segnali multimediali", e per ottimi motivi, quelli destinati a organi di senso basati sul contatto: tatto, odorato e gusto.)

Questo è avvenuto sopratutto per la progressiva unificazione, delle tecnologie di produzione e diffusione di questi segnali, in due sole tecnologie: la registrazione digitale (audio, video e videoscrittura) e la comunicazione telematica.

Il corso "Laboratorio di informatica multimediale" si propone di dare agli studenti gli strumenti di base necessari a capire, utilizzare e sperimentare queste due tecnologie.

Argomenti trattati ed esercitazioni di laboratorio:

- tecnologie di comunicazione via rete
  - comunicazione uomo-macchina
  - comunicazione macchina-macchina
  - comunicazione uomo-uomo
    - sincrona e asincrona
    - push e pull
    - uno a uno, uno a molti, molti a molti, molti a uno
  - tecniche d' uso efficace della rete.
- codifica di (iper) testi
  - rappresentazione dei dati simbolici

- ASCII, ASCII esteso, Unicode e codifiche
- HTML
  - storia e versioni
  - conformance e verifica
  - accessibilità e verifica
- uso della grafica
- uso efficace del colore

- registrazione digitale
  - [Argomenti omessi]

Programma svolto

NB: ogni “punto” si riferisce ad un incontro in aula (2 ore di lezione)

1. 16 marzo 2011

Indagine sulla disponibilità fra gli studenti di strumenti di elaborazione e comunicazione in rete (soprattutto e-mail); configurazione lista di distribuzione LabInfoMM2010-2011.

Esposizione degli strumenti di teleLaboratorio a disposizione del corso:

http://trusso.freeshell.org/Architettura2010-2011/Strumenti/Architettura.htm

Comunicazione uomo-macchina: concetto di sessione TCP, client-server, esempio di connessioni telnet a un server Web e a un server mail:

```
C:> telnet
telnet> open www.google.it 80
GET / HTTP/1.1
Host: %s

HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html
Set-Cookie:
PREF=ID=3086695e93534ecd:TM=1129481453:LM=1129481453:S=37vo5bfnF2fabCYg;
expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/; domain=.google.com
Server: GWS/2.1
Transfer-Encoding: chunked
Date: Sun, 16 Oct 2005 16:50:53 GMT

9db
<html><head><meta http-equiv="content-type" content="text/html; charset=ISO-
8859-1"><title>Google</title><style><!--
body,td,a,p,.h{font-family:arial,sans-serif;}
.h{font-size: 20px;}
.q{color:#0000cc;}
//-->
</style>
<script>
```

```
<!--
function sf(){document.f.q.focus();}
// -->
</script>
</head><body bgcolor=#ffffff text=#000000 link=#0000cc vlink=#551a8b
alink=#ff0000 onLoad=sf() topmargin=3 marginheight=3><center> ecc. ecc.



telnet> open mail.tin.it 25
Trying 62.211.72.20...
Connected to mail.tin.it (62.211.72.20).
Escape character is '^]'.
220 vsmtp14.tin.it ESMTP Service (7.2.060.1) ready
HELO whitehouse.gov
250 vsmtp14.tin.it
MAIL FROM: <bush@whitehouse.gov>
250 MAIL FROM:<bush@whitehouse.gov> OK
RCPT TO:trusso@tin.it
501 Syntax error in parameters or arguments to RCPT command
RCPT TO:<trusso@tin.it>
250 RCPT TO:<trusso@tin.it> OK
DATA
354 Start mail input; end with <CRLF>.<CRLF>
From: George Bush
TO: Tony Blair
Subject: news about iraqi petroleum

Tony, call me as soon as possible-Kofi Annan has just told me something...
George
.
250 <4336E405008D3067> Mail accepted
```

Connessione ad un server di accesso remoto alla shell (“telnet” nel senso comune del termine):

```
telnet> open otaku.freeshell.org 23
Trying 192.94.73.2...
Connected to otaku.freeshell.ORG (192.94.73.2).
Escape character is '^]'.

sdf.lonestar.org (ttyr1)
if new, login 'new' ..

login: trusso
Password:
Last login: Sun Oct 16 15:02:54 2005 from host127-171.pool8260.interbusiness.it
on ttypb
Copyright (c) 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004
    The NetBSD Foundation, Inc.  All rights reserved.
Copyright (c) 1982, 1986, 1989, 1991, 1993
    The Regents of the University of California.  All rights reserved.

You have mail.
you have 2 pending notifications
type 'notify -r' to retrieve them
$
```

Primi comandi essenziali Unix: vedi

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/daStudiare/001-ComandiUnix.pdf

Esercitazione in aula (e chi non può, a casa):
aprire un telnet sulla macchina otaku
posizionarsi nella directory `<home>LabinfoMM2010-2011/`
creare una directory personale *CognomeNo*

2. 23 marzo 2011

Altri comandi essenziali Unix: completato

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/daStudiare/001-ComandiUnix.pdf

`ls, ls -l, ls-la, cd, pwd`: **viste** da "terminale nero" sul system file del sistema remoto

Altra possibile vista: **FTP**

Client grafico per windows (Filezilla). Scaricabile da

http://filezilla-project.org/download.php?type=client

Configurazione di Filezilla: didatticamente, impostato cosi':

Modifica – impostazioni – Trasferimenti – Tipi di file

eliminate tutte le estensioni che verrebbero trasferite in modalità "ASCII": in questo modo, TUTTI i file vengono trasmessi "bit per bit", in formato BINARIO.

Prove di trasmissione di file creati su Windows e trasmessi su server Unix in binario, e viceversa:

```
$ hexdump -C pippo
00000000  70 69 70 70 6f 0a 70 6c  75 74 6f 0a 70 61 70 65  |pippo.pluto.pape|
00000010  72 69 6e 6f 0a 70 69 70  70 6f 20 70 6c 75 74 6f  |rino.pippo pluto|
00000020  20 70 61 70 65 72 69 6e  6f 0a                    | paperino.|
0000002a
```

Inviato su Windows, il notepad lo visulalizza come

```
pippo?pluto?paperino?pippo pluto paperino?
```

```
Stesso file creato su Windows e inviato a Unix:
```

```
$ hexdump -C uno.txt
00000000  70 69 70 70 6f 0d 0a 70  6c 75 74 6f 0d 0a 70 61  |pippo..pluto..pa|
00000010  70 65 72 69 6e 6f 0d 0a  70 69 70 70 6f 20 70 6c  |perino..pippo pl|
00000020  75 74 6f 20 70 61 70 65  72 69 6e 6f 0d 0a        |uto paperino..|
$
```

Interpretazione dei caratteri ASCII e dei caratteri speciali di fine record **0a** e **0d,** e cenni sui caratteri di controllo usati per le telescriventi (tty) :

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/consultazione/001-ASCII_Chart.pdf

Perchè Windows usa 0a **E** 0d (stampanti stupide), mentre Unix usa solo 0a.

Caratteri “speciali” (con accenti ed altri segni diacritici d' uso nazionale):

aggiungendo su Windows una riga

èéòçà°ù§ì£$$

```
$ hexdump -C uno.txt
00000000  70 69 70 70 6f 0d 0a 70  6c 75 74 6f 0d 0a 70 61  |pippo..pluto..pa|
00000010  70 65 72 69 6e 6f 0d 0a  70 69 70 70 6f 20 70 6c  |perino..pippo pl|
00000020  75 74 6f 20 70 61 70 65  72 69 6e 6f 0d 0a e8 e9  |uto paperino....|
00000030  f2 e7 e0 b0 f9 a7 ec a3  24                       |........$|
00000039
```

Perchè non è possibile controllare come verranno visualizzati i caratteri > 127:

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/consultazione/003-wikipedia.org-ISO_8859.html

3. 30 marzo 2011

Discussione sul significato di Multimedialità

Altri utenti sullo stesso sistema, come comunicare con essi: `talk`

Concetto base della comunicazione fra utenti connessi ad uno stesso sistema: **l’altro** deve avere il permesso di scrivere su un **proprio** file (stdout, mailbox).

Vista generale sulle comunicazioni in rete:
http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/daStudiare/002-NONWEB102001.HTM

Spiegazione di termini:

- uno a uno, uno a molti, molti a molti, molti a uno
- sincrona e asincrona
- push e pull

Tecnica fondamentale della comunicazione corretta:

- usare la comunicazione push solo per brevi segnalazioni di novità e di disponibilità di altro materiale
- mettere a disposizione il materiale più ingombrante in modo che sia ottenibile con tecnologia pull

- nei messaggi push inserire i link al materiale disponibile in pull

Comunicazioni macchina-macchina: routing, traduzioni nomi a dominio.

Strumenti di diagnosi e controllo: traceroute, ping, nslookup

Ping: Significato dei campi di output e in particolare del TTL.
Traceroute
- dove sta Otaku?
- Quanto dista Trieste da Capodistria?

4. 6 aprile 2011

Gruppi di discussione
Strumenti di telecollaborazione e teledidattica

5. 13 aprile 2011

HTML. Scrittura di pagine web con un semplice text editor.

6. 20 aprile 2011

Validazione W3C HTML
Uso intelligente di motori di ricerca

7. 4 maggio 2011

Inizio seconda parte: Assembler

Seconda parte: assembler 8088 e 80386

Argomenti di questi complementi sono il capitolo 7 e l' appendice C del testo “Structured Computer Organization” di Andrew S. Tanenbaum, quinta edizione. Alcuni esempi per le esercitazioni sono tratte dal capitolo 5; altri sono scritti ex novo.

Programma svolto

NB: ogni “punto” si riferisce ad un incontro in aula (2 ore di lezione)

1. **4 maggio 2011**

Esposizione tools usabili:

Materiale didattico (però su piattaforma Aplha e non Intel; gcc configurato per emettere codice Assembly Alpha, non Intel):

```
telnet otaku.freeshell.org
user: trusso
pw:
```
(comunicata a voce)

Chi non dispone di un PC Linux con gcc può usare il seguente, per gentile concessione dell' azienda proprietaria (Enteos srl):

```
ssh -l trusso enteos2.area.trieste.it
(user: trusso)
pw:
```
(la stessa di otaku)

oppure, in caso di emergenza, il server domestico

```
ssh -l trusso blacky.terra32.net
(user: trusso)
pw:
```
(la stessa di otaku)

Attenzione: enteos2 e blacky non accettano per motivi di sicurezza il telnet, è necessario usare ssh: su Windows si può usare putty, o ssh a finestre.

Per le esercitazioni è necessario l' interprete per l' assembler **as88** che si trova nel CD allegato al Tanenbaum, si trova anche qui:

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/Esercitazioni/8088_tra/

in versione Windows, Solaris e Linux.
Leggere il file readme.txt e poi installare la versione voluta.

ASSEMBLER **in italiano** significa 2 cose che in inglese hanno nomi diversi:

ASSEMBLY LANGUAGE

- Linguaggio di programmazione in cui a ogni istruzione simbolica corrisponde esattamente una istruzione macchina
- Architetture diverse hanno sicuramente linguaggi diversi
- Possone esistere, *ed esistono*, diversi linguaggiper la stessa architettura
  - es: per 80386 e successivi: sintassi ATT e sintassi Intel, con e senza prefissi

ASSEMBLY LANGUAGE COMPILER (In short, **ASSEMBLER**)

- E' un *programma* che accetta in input files contenenti programmi scritti in assembly language e produce in output files contenenti gli stessi programmi in linguaggio macchina. Lo scopo finale e' quello di ottenere un ***file eseguibile da un certo sistema operativo***: cosa che di norma l'assembler non produce. Il programma che effettua questo ultimo passo si chiama ***linker*** (o *loader* o *mapper).*

  Per ogni architettura, e per ogni assembly language definito per essa, deve esistere almeno un compilatore assembler in grado di *tradurre* l' assembly language in istruzioni macchina.

  Un compilatore, essendo un programma, e' anch'esso stato scritto: in assembly language, o piu' spesso in **c** o altri linguaggi superiori, e poi compilato e linkato.

*E come e' stato scritto il **primo** assembler?*

*E' stato scritto in assembly language etradotto **a mano** in linguaggio macchina. Sì, si può; e il primo assembly language era abbastanza semplice.*

Il secondo assembler è stato compilato con il primo. E via avanti.

Quelli odierni vengono compilati con i compilatori già esistenti.

Catena della compilazione per un linguaggio:

- traduzione da linguaggio superiore ad Assembly
- traduzione da Assembly a binario rilocabile con External e Undefined
- linking di più rilocabili in un eseguibile

Il NOSTRO compilatore: **gcc.**

gcc != gnu c compiler
gcc = gnu compiler complex

include, o chiama: traduttore c->assembly, traduttore fortran->c, assembler, linker.

Esempio di partenza:

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/Esercitazioni/CompileLink/addintmain.c	http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/Esercitazioni/CompileLink/addintsubs.c

`#include <stdio.h>`
 	
`int main ()`
`  {`
`  int uno;`
`  int due;`
`  int tre;`
 	
`  uno = 257;`
`  due = 64;`
`  tre = addint (uno,due);`
`  printf ("%d + %d = %d \n" , uno, due, tre);`
`  tre = multint (uno,due);`
`  printf ("%d * %d = %d \n" , uno, due, tre);`
`  }`
	`int addint(int a, int b)`
	`{`
	`int c;`
	`c = a+b;`
	`return c;`
	`}`
	
	`int multint(int a, int b)`
	`{`
	`int c;`
	`c = a * b;`
	`return c;`
	`}`

Esecuzione della catena passo passo:

I passo.1) Traduzione da C in Assembly per piattaforma Intel Pentium con sintassi Intel con il comando (Su macchina Intel con S.O. Linux)

```
gcc -masm=intel -m32 -mno-accumulate-outgoing-args
-mpreferred-stack-boundary=2 -S addintsubs.c -o addintsubs.s
```

Spiegazione delle opzioni:

`-masm=intel`
genera un programma in assembly language con sintassi intel anziche' ATT (default)

`-m32`
traduci in assembly language a 32 bit anche se la macchina su cui si sta lavorando ha architettura a 64 bit, e quindi il default del gcc è `-m64`

`-mno-accumulate-outgoing-args -mpreferred-stack-boundary=2`
NON generare codice ottimizzato, che e' piu' veloce ma maschera la semplice struttura dell'uso dello stack

`-S`
fermati dopo la traduzione da c ad assembly language

`-o`
quello che segue è il nome del file di output.

I passo.2) Traduzione da C in Assembly per piattaforma Intel Pentium con sintassi di default (AT&T) con il comando (Su macchina Intel con S.O. Linux)

```
gcc  -m32 -mno-accumulate-outgoing-args -mpreferred-stack-boundary=2 -S addintsubs.c -o addintsubs.s
```

Stesso di prima, ma senza l'opzione `-masm=intel`

I passo.3) Traduzione da C in Assembly per piattaforma Alpha con sintassi di default (AT&T) con il comando (Su macchina Alpha con S.O. Linux)

```
gcc  addintsubs.c -S  -o addintsubs.s
```

II passo) assembling dei risultati con il comando

```
as -a addintsubs.s -o addintsubs.o >  addintsubs.list
```

quello che vene generato è un file contenete il programma RILOCABILE (non ancora quello eseguibile.

III passo) produzione del file eseguibile o linkato con il **linker** o **loader** standard, *ld*

```
ld  addintsubs.o addintmain.o /usr/lib/crtn.o -lc -o addint.eseguibile
```

Oppure, *molto più semplicemente*, con LO STESSO risultato:

```
gcc  addintsubs.o addintmain.o -o addint.eseguibile
```

(E' gcc che si preoccupa di chiamare ld con le opzioni corrette.)

IVpasso) esecuzione del file eseguibile prodotto:

```
./addint.eseguibile
```

("./" serve perche' di solito la directory corrente non è fra quelle ispezionate)

- o -

La stessa catena viene eseguita con un'unico comando da gcc

```
gcc -m32 addintmain.c addintsubs.c -o addint.eseguibile
./addint.eseguibile
```

(E' gcc che si preoccupa di chiamare ld con le opzioni corrette.)

E questo comando, utilissimo, produce un listato con le istruzioni c, l'equivalente in assembly in cui sono state tradotte, e la compilazione in codice binario:

```
gcc -m32 -mno-accumulate-outgoing-args -mpreferred-
stack-boundary=2 -c -g -Wa,-a,-ad   addintmain.c
addintsubs.c         -o addint.eseguibile > file.lista
```

tutto in un'unica lista.

Confronto dei listati assembler:

Esempi di listato assembler

1)

```
GAS LISTING addintsubs.s                        page 1


   1                            .file   "addintsubs.c"
   2                            .intel_syntax
   3                            .text
   4                    .globl addint
   5                            .type   addint, @function
   6                    addint:
   7 0000 55                    push    %ebp
   8 0001 89E5                  mov     %ebp, %esp
   9 0003 83EC04                sub     %esp, 4
  10 0006 8B450C                mov     %eax, DWORD PTR [%ebp+12]
  11 0009 034508                add     %eax, DWORD PTR [%ebp+8]
  12 000c 8945FC                mov     DWORD PTR [%ebp-4], %eax
  13 000f 8B45FC                mov     %eax, DWORD PTR [%ebp-4]
  14 0012 C9                    leave
  15 0013 C3                    ret
  16                            .size   addint, .-addint
  17                    .globl multint
  18                            .type   multint, @function
  19                    multint:
  20 0014 55                    push    %ebp
  21 0015 89E5                  mov     %ebp, %esp
  22 0017 83EC04                sub     %esp, 4
  23 001a 8B4508                mov     %eax, DWORD PTR [%ebp+8]
  24 001d 0FAF450C              imul    %eax, DWORD PTR [%ebp+12]
  25 0021 8945FC                mov     DWORD PTR [%ebp-4], %eax
  26 0024 8B45FC                mov     %eax, DWORD PTR [%ebp-4]
  27 0027 C9                    leave
  28 0028 C3                    ret
  29                            .size   multint, .-multint
  30                            .section        .note.GNU-stack,"",@progbits
  31                            .ident  "GCC: (GNU) 3.3.3 20040412 (Red Hat
Linux 3.3.3-7)"
GAS LISTING addintsubs.s                        page 2
```

Esempi di listato assembler

```
DEFINED SYMBOLS
                           *ABS*:00000000 addintsubs.c
        addintsubs.s:6     .text:00000000 addint
        addintsubs.s:19    .text:00000014 multint

NO UNDEFINED SYMBOLS
```

2)

```
GAS LISTING addintsubs.s                        page 1


   1                            .file   "addintsubs.c"
   2                            .text
   3                    .globl addint
   4                            .type   addint, @function
   5                    addint:
   6 0000 55                    pushl   %ebp
   7 0001 89E5                  movl    %esp, %ebp
   8 0003 83EC04                subl    $4, %esp
   9 0006 8B450C                movl    12(%ebp), %eax
  10 0009 034508                addl    8(%ebp), %eax
  11 000c 8945FC                movl    %eax, -4(%ebp)
  12 000f 8B45FC                movl    -4(%ebp), %eax
  13 0012 C9                    leave
  14 0013 C3                    ret
  15                            .size   addint, .-addint
  16                    .globl multint
  17                            .type   multint, @function
  18                    multint:
  19 0014 55                    pushl   %ebp
  20 0015 89E5                  movl    %esp, %ebp
  21 0017 83EC04                subl    $4, %esp
  22 001a 8B4508                movl    8(%ebp), %eax
  23 001d 0FAF450C              imull   12(%ebp), %eax
  24 0021 8945FC                movl    %eax, -4(%ebp)
  25 0024 8B45FC                movl    -4(%ebp), %eax
  26 0027 C9                    leave
  27 0028 C3                    ret
  28                            .size   multint, .-multint
  29                            .section        .note.GNU-stack,"",@progbits
  30                            .ident  "GCC: (GNU) 3.3.3 20040412 (Red Hat
Linux 3.3.3-7)"
GAS LISTING addintsubs.s                        page 2


DEFINED SYMBOLS
                           *ABS*:00000000 addintsubs.c
        addintsubs.s:5     .text:00000000 addint
        addintsubs.s:18    .text:00000014 multint

NO UNDEFINED SYMBOLS
```

3)

```
GAS LISTING addintsubsalpha.s                   page 1


   1                            .set noat
   2                            .set noreorder
   3                            .set nomacro
```

Esempi di listato assembler

```
   4                            .text
   5                            .align 2
   6                            .globl addint
   7                            .ent addint
   8                    $addint..ng:
   9                    addint:
  10                            .frame $15,32,$26,0
  11                            .mask 0x4008000,-32
  12 0000 E0FFDE23              lda $30,-32($30)
  13 0004 00005EB7              stq $26,0($30)
  14 0008 0800FEB5              stq $15,8($30)
  15 000c 0F04FE47              bis $31,$30,$15
  16                            .prologue 0
  17 0010 0104F047              mov $16,$1
  18 0014 0204F147              mov $17,$2
  19 0018 10002FB0              stl $1,16($15)
  20 001c 14004FB0              stl $2,20($15)
  21 0020 10004FA0              ldl $2,16($15)
  22 0024 14002FA0              ldl $1,20($15)
  23 0028 01004140              addl $2,$1,$1
  24 002c 18002FB0              stl $1,24($15)
  25 0030 18002FA0              ldl $1,24($15)
  26 0034 0100E143              addl $31,$1,$1
  27 0038 0004E147              mov $1,$0
  28 003c 1E04EF47              mov $15,$30
  29 0040 00005EA7              ldq $26,0($30)
  30 0044 0800FEA5              ldq $15,8($30)
  31 0048 2000DE23              lda $30,32($30)
  32 004c 0180FA6B              ret $31,($26),1
  33                            .end addint
  34                            .align 2
  35                            .globl multint
  36                            .ent multint
  37                    $multint..ng:
  38                    multint:
  39                            .frame $15,32,$26,0
  40                            .mask 0x4008000,-32
  41 0050 E0FFDE23              lda $30,-32($30)
  42 0054 00005EB7              stq $26,0($30)
  43 0058 0800FEB5              stq $15,8($30)
  44 005c 0F04FE47              bis $31,$30,$15
  45                            .prologue 0
  46 0060 0104F047              mov $16,$1
  47 0064 0204F147              mov $17,$2
  48 0068 10002FB0              stl $1,16($15)
  49 006c 14004FB0              stl $2,20($15)
  50 0070 10004FA0              ldl $2,16($15)
  51 0074 14002FA0              ldl $1,20($15)
  52 0078 0100414C              mull $2,$1,$1
  53 007c 18002FB0              stl $1,24($15)
  54 0080 18002FA0              ldl $1,24($15)
  55 0084 0100E143              addl $31,$1,$1
  56 0088 0004E147              mov $1,$0
  57 008c 1E04EF47              mov $15,$30
GAS LISTING addintsubsalpha.s                   page 2


  58 0090 00005EA7              ldq $26,0($30)
  59 0094 0800FEA5              ldq $15,8($30)
  60 0098 2000DE23              lda $30,32($30)
```

Esempi di listato assembler

```
  61 009c 0180FA6B              ret $31,($26),1
  62                            .end multint
  63                            .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3
20040520)"
GAS LISTING addintsubsalpha.s                   page 3


DEFINED SYMBOLS
   addintsubsalpha.s:9      .text:0000000000000000 addint
   addintsubsalpha.s:38     .text:0000000000000050 multint

NO UNDEFINED SYMBOLS
```

Considerazioni :
1 e 2 assembly language diversi, portano allo stesso codice binario;
3 linguaggio assembly completamente diverso, codice binario totalmente diverso, architettura con istruzioni binarie tutte della stessa lunghezza (alpha è RISC!, Intel è principalmente backward compatibile e quindi spesso inutilmente complicato).

2. 5 maggio 2011

Numero e funzioni dei registri 8088 (Tanenbaum appendice C) **e Pentium** (Tanenbaum cap. 5)

Program counter (PC o IP): necessita' che l'istruzione in esecuzione venga decodificata prima di incrementare PC (per saperne la lunghezza)

Funzionamento dello stack

Condition code, status flags più importanti (Zero, Sign(=negative), Overflow, Carry)

3. 11 maggio 2011

Il Flag I del CC

Significato di "interrupt"

Esecuzione di un'istruzione ad un indirizzo assoluto *diverso* da quello ottenuto sommando al PC il code segment register * 4. Il bit I del CC viene settato a 0 (other interrupts prevented)

Prima istruzione delle routine di interrupt

E' una call alla routine di gestione: il questo modo il PC viene salvato nello stack, e la routine di gestione interrupt potra' terminare con una RET (o con una IRET, che anche setta il bit I del CC a 1)

Preprocessing e postprocessing

Dopo aver salvato tutto quello che dovrà usare, la routine puo' eseguire un set del bit I e proseguire in "user mode" (Interruptable). In questo caso finisce con una RET.

Istruzione di chiamata al supervisore (`int, sys, er....`)

GENERA un interrupt a una locazione fissa. In questo modo un programma può chiedere un servizio al sistema operativo (p.es. I/O) che da solo non ha il permesso di eseguire.

Esame di un programma assembler (il solito "hello world"):

```
! Simple "hello world" program
! See section 9.8.1.

     _EXIT     = 1        !  1
     _WRITE    = 4        !  2
     _STDOUT   = 1        !  3
.SECT .TEXT               !  4
start:                    !  5
     MOV  CX,de-hw        !  6
     PUSH CX              !  7
     PUSH hw              !  8
     PUSH _STDOUT         !  9
     PUSH _WRITE          ! 10
     SYS                  ! 11
     ADD  SP,8            ! 12
     SUB  CX,AX           ! 13
     PUSH CX              ! 14
     PUSH _EXIT           ! 15
     SYS                  ! 16
.SECT .DATA               ! 17
hw:                       ! 18
.ASCII    "Hello World\n" ! 19
de:  .BYTE     0          ! 20
.SECT .BSS
```

Uso di **`t88`** (Da CD Tanenbaum; descritto in **Tanenbaum C.6**) per eseguire questo programma
Versione `printf / int 0x80` per pentium 32 bit:

```
#****************************************************
        .text
#****************************************************
     .globl main
     .type main, @function
main:

     push      $stringmsg
     call      printf
     addl      $4, %esp

     movl $asciimsg, %ecx
     movl $asciimsglen, %edx
     call PrintString
exit:
     movl $0,%ebx        # exit code
     movl $1,%eax        # sys_exit
     int  $0x80          # sys call

PrintString:   #
     #  INPUT: ecx=address of buffer, edx=buffer lenght
        push   %eax
     push %ebx
     movl $1, %ebx        # write to stdout
     mov  $4, %eax       # write to file handle
     int  $0x80          # ignore return value
     pop  %ebx
     pop  %eax
     ret            #
#****************************************************
     .data
#****************************************************

stringmsg:     .string "ciao mondo!\n"

asciimsg: .ASCII "di nuovo ciao!\n"
asciimsglen    = . - asciimsg
```

problemi che può dare l’uso nello stesso programma di entrambe le tecniche.

4. 12 maggio 2011

direttive al compilatore:

Sezioni `.SECT .TEXT, .DATA, .BSS`

Nomi simbolici `(es. _WRITE=4, _STDOUT = 1)`

Labels

Istruzioni: esame della documentazione, reperibile nell' appendice C del Tanenbaum. E in:

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/consultazione/ManualiIntel/

(Elenco delle istruzioni Pentium nei manuali 2A e 2B).

Indirizzamento ammesso nelle istruzioni 8088 per destinazione e source
Operando immediato (source only), indirizzamento diretto, indirizzamento indiretto tramite registro, registro con indice e dislocamento

(Tanenbaum cap. C.3.2, figura C.3 e descrizione nelle stesse pagine)

Tabella semplificativa (as del gcc in sintassi AT&T):

Indirizzamento	**costanti**	**Label (es. `stringa:`)**	**Registro**
Immediato (lui, proprio lui)	**`$20`** e' proprio il numero 20 NB: l'AS88 usa qui invece solo **`20`**	**`$stringa`** L'indirizzo effettivo (RILOCATO DAL LINKER) dove compare la label "stringa:"	--
diretto Il dato che lui contiene e' l'operando	**`20`** il dato contenuto in memoria a partire dall'indirizzo effettivo 20 (NON USARE!) NB: l'AS88 usa qui invece **`(20)`**	**`stringa`** il dato contenuto in memoria a partire dall'indirizzo (RILOCATO DAL LINKER) dove compare la label "stringa:"	**`%ebx`** il dato contenuto nel registro
Indiretto il dato che lui contiene è usato per calcolare un indirizzo di memoria dove si trova l'operando	--	--	**`(%ebx)`** il dato contenuto in memoria a partire dall'indirizzo effettivo contenuto nel registro OPPURE **`+-20(%esi)(%ebp)`** e altre combinazioni: i contenuti e il dato immediato vengono sommati insieme per dare l'indirizzo effettivo dive si trova l'operando

Analisi sistematica delle principali istruzioni 8088
(Tanenbaum cap. C.4, figura C.4)

con approfondimenti (su architettura a 16 e 32 bit) sulla documentazione ufficiale Intel:

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/consultazione/ManualiIntel/INTEL-2A-253666.pdf

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/consultazione/ManualiIntel/INTEL-2B-253666.pdf

5. 18 maggio 2011

Necessità del doppio passaggio Assembler per la creazione della Symbol Table

Programmazione strutturata in Assembler: chiamata a subroutines

Analisi della traduzione in assembly del gcc delle chiamate a subroutine. Materiale: programma c di esempio:

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/Esercitazioni/ChiamateSubInC/TestCall.c

```
#include <stdio.h>
void fanulla ()
  {    return;  }

int dammizero ()
  {   return 0;  }

int addint2 (int a, int b )
  { return a+b;  }

int addint3 (int a, int b, int c )
  { return a+b+c;  }

int addint4 (int a, int b, int c, int d )
  { return a+b+c+d;  }

int addint5 (int a, int b, int c, int d, int e )
  { /* questa routine la complichiamo un po' */
    int somma, somma2, somma3;
    somma = a+b+c+d+e;
    somma2 = addint2 (somma, a);
    somma3 = addint3 (somma, a, b);

    printf ("somma: %d somma2: %d somma3: %d\n", somma, somma2,
somma3);
```

```
          return somma;  }

      int main ()
        {
        int uno;
        int due;
        int tre;
        int quattro;
        int cinque;
        int sei;
        uno = 257;
        fanulla;
        due = dammizero();
        tre = addint2 (uno,due);
        quattro = addint3 (uno,due,tre);
        cinque = addint4 (uno,due,tre,quattro);
        sei = addint5 (uno,due,tre,quattro,cinque);
        printf ("uno: %d due: %d tre: %d quattro: %d cinque: %d sei:
      %d \n" , uno, due, tre, quattro, cinque, sei);
        printf("addio mondo crudele\n");
        }
```

Compilato in Assembly language, commentato e modificato:

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/Esercitazioni/ChiamateSubInC/TestCall.c.s.commented

```
      .file "TestCall.c"
      .text
.globl fanulla
      .type fanulla, @function
fanulla:
      pushl %ebp
      movl  %esp, %ebp
# la routine non fa assolutamente niente

      popl  %ebp

# ATTENZIONE: la sequenza "standard" per ripristinare
#  la situazione precedente dovrebbe contenere DUE istruzioni:
#           movl  %ebp, %esp
#           popl  %ebp
# che sono "la chiusura delle parentesi" aperte dalle prime
#   due istruzioni che seguono la label "fanulla:".
# Il GCC non ha messo la movl %ebp, %esp perche' si e' accorto
# che il registro esp non e' MAI stato utilizzato.
#
# NON fatelo anche voi!  E' PERICOLOSO!
      ret
      .size fanulla, .-fanulla

#-----------------fine routine-------------
```

```
.globl dammizero
      .type dammizero, @function
dammizero:
      pushl %ebp
      movl  %esp, %ebp
      movl  $0, %eax

# convenzione del c (da adottare SEMPRE):
# le funzioni (non void) restituiscono il risultato in eax.

      popl  %ebp
# Come sopra:
# Il GCC non ha messo la movl %ebp, %esp perche' si e' accorto
# che il registro esp non e' MAI stato utilizzato.
# NON fatelo anche voi!  E' PERICOLOSO!

      ret
      .size dammizero, .-dammizero

#------------------fine routine--------------

.globl addint2
      .type addint2, @function
addint2:
      pushl %ebp
      movl  %esp, %ebp
      movl  12(%ebp), %eax
#    come dire: %ebp + 3*4, ossia 3*"Architetture/8"
#    cioe' PENULTIMO argomento messo in push
#    (ossia SECONDO nel C)
      movl  8(%ebp), %edx
#    come dire: %ebp + 2*4, ossia 2*"Architetture/8"
#    cioe' ULTIMO argomento messo in push
#    (ossia PRIMO nel C)

# PERCHE'??
# Perche' %ebp + 0*4 contiene il PRECEDENTE VALORE di %ebp
# e  %ebp + 1*4 contiene l' indirizzo di ritorno
# (push implicito fatto dalla call; pop implicito dalla ret)

      leal  (%edx,%eax), %eax

# da notare lo strano uso dell'istruzione LEA per la somma.
# poteva essere invece:       addl  8(%ebp), %eax
# senza la MOV che la precede.
#
# a questo punto eax contiene il risultato finale,
# non resta niente da fare:  chiudere parentesi e return.

      popl  %ebp
# Come sopra:
# Il GCC non ha messo la movl %ebp, %esp perche' si e' accorto
# che il registro esp non e' MAI stato utilizzato.
# NON fatelo anche voi!  E' PERICOLOSO!
      ret
      .size addint2, .-addint2

#------------------fine routine--------------
```

```
.globl addint3
      .type addint3, @function
addint3:
      pushl %ebp
      movl  %esp, %ebp
      movl  12(%ebp), %eax
      movl  8(%ebp), %edx
      leal  (%edx,%eax), %eax
      addl  16(%ebp), %eax
      popl  %ebp
      ret
      .size addint3, .-addint3

#------------------fine routine--------------

.globl addint4
      .type addint4, @function
addint4:
      pushl %ebp
      movl  %esp, %ebp
      movl  12(%ebp), %eax   # SECONDO argomento della call
      movl  8(%ebp), %edx    # PRIMO argomento della call
      leal  (%edx,%eax), %eax
      addl  16(%ebp), %eax   # TERZO argomento della call
      addl  20(%ebp), %eax   # QUARTO argomento della call
# da notare l'uso dell'istruzione LEA SOLO per la PRIMA somma.
# poteva essere invece:       addl  8(%ebp), %eax
# senza la MOV che la precede.
# Poi invece viene usata la normale ADD.

      popl  %ebp

# Come sopra:
# Il GCC non ha messo la movl %ebp, %esp perche' si e' accorto
# che il registro esp non e' MAI stato utilizzato.

      ret
      .size addint4, .-addint4

#------------------fine routine--------------

      .section    .rodata
      .align 4
.LC0:
      .string     "somma: %d somma2: %d somma3: %d\n"
      .text
.globl addint5
      .type addint5, @function
addint5:
      pushl %ebp
      movl  %esp, %ebp
# questa sopra e' la sequenza standard "Aperta parentesi",
# cioe' salva esp e inizializza ebp in modo da avere visibili gli argomenti

#     subl  $24, %esp
# questa invece e' NUOVA: a cosa serve?
# RISERVA 24 byte (= 6 parole da 32 bit) nello stack
# per ospitarvi variabili LOCALI.
# problemi di allineamento?
# proviamo a riservare spazio solo per tre:
      subl  $12, %esp
```

```
# (funziona lo stesso)

      movl  12(%ebp), %eax  # SECONDO argomento della call (b)
      movl  8(%ebp), %edx     # PRIMO argomento della call (a)
      leal  (%edx,%eax), %eax
      addl  16(%ebp), %eax    # TERZO argomento della call (c)
      addl  20(%ebp), %eax    # QUARTO argomento della call (d)
      addl  24(%ebp), %eax    # QUINTO argomento della call (e)

# Qui la cosa si fa interessante: DOVE viene messo il risultato
#  di  somma = a+b+c+d+e;?

      movl  %eax, -12(%ebp)
# evidentemente ebp - 12 punta a "somma", nello spazio RISERVATO
#                                         alle variabili locali

      pushl 8(%ebp)    # a (fra gli argomenti)
      pushl -12(%ebp)  # somma (nelle variabili locali)
      call  addint2
      addl  $8, %esp     # butta via gli argomenti nello stack
      movl  %eax, -16(%ebp)
# era     somma2 = addint2 (somma, a);
# evidentemente ebp - 16 punta a "somma2", nello spazio RISERVATO

      pushl 12(%ebp)   # a (fra gli argomenti)
      pushl 8(%ebp)    # b (fra gli argomenti)
      pushl -12(%ebp)  # somma (nelle variabili locali)
      call  addint3
      addl  $12, %esp     # butta via gli argomenti nello stack
      movl  %eax, -20(%ebp)
# era     somma3 = addint3 (somma, a, b);
# evidentemente ebp - 20 punta a "somma3", nello spazio RISERVATO.
# infatti, nella chiamata
#   printf ("somma: %d somma2: %d somma3: %d\n", somma, somma2, somma3);:

      movl  $.LC0, %eax   # indirizzo della stringa in eax
      pushl -20(%ebp)     # somma3
      pushl -16(%ebp)     # somma2
      pushl -12(%ebp)     # somma  (nota l'ordine inverso)
      pushl %eax          # ultimo (ossia primo) argomento: stringa
      call  printf
      addl  $16, %esp     # butta via gli argomenti nello stack
# fine chiamata a printf

 # return somma viene fatto qui:
      movl  -12(%ebp), %eax

#     leave
# l'istruzione "leave" esegue proprio, in un'unica istruzione,
# la sequenza standard di "chiusura parentesi" che ci sostituiamo:
            movl  %esp, %ebp
            popl  %ebp
#  che qui e' necessaria perche' ANCHE il registro esp e' stato usato
#  per riservare variabili locali, e
#  in ALTRE chiamate a subroutines.
# fine sostituzione
      ret
      .size addint5, .-addint5

#-----------------fine routine-------------
```

```
	.section    .rodata
	.align 4
.LC1:
	.string     "uno: %d due: %d tre: %d quattro: %d cinque: %d sei: %d \n"
.LC2:
	.string     "addio mondo crudele"
	.text
.globl main
	.type main, @function
main:
	leal  4(%esp), %ecx
	andl  $-16, %esp
# fin qui e' solo un allineamento dello stack, INUTILE.
	pushl -4(%ecx)
# questo salva proprio il top dello stack PRIMA dell'allineamento.
# ma comporta un push: quindi alla fine ci sara' un pop

	pushl %ebp
	movl  %esp, %ebp
# questa e' la vera "parentesi aperta"

	pushl %ecx
#anche questo e' un salvataggio di ottimizzazione

	subl  $36, %esp       # riserva spazio per 9 variabili locali
                              # (ma ne basterebbero 6)

	movl  $257, -12(%ebp)  # il famoso 257 va in "uno"
	call  dammizero
	movl  %eax, -16(%ebp) # zero va in "due"

	pushl -16(%ebp)      # due
	pushl -12(%ebp)      # uno
	call  addint2
	addl  $8, %esp       # pulisci lo stack
	movl  %eax, -20(%ebp) # risultato in "tre"

	pushl -20(%ebp)     # tre
	pushl -16(%ebp)     # due
	pushl -12(%ebp)     # uno
	call  addint3
	addl  $12, %esp       # pulisci lo stack
	movl  %eax, -24(%ebp) # risultato in "quattro"

	pushl -24(%ebp)     # quattro
	pushl -20(%ebp)     # tre
	pushl -16(%ebp)     # due
	pushl -12(%ebp)     # uno
	call  addint4
	addl  $16, %esp       # pulisci lo stack
	movl  %eax, -28(%ebp) # risultato in "cinque"

	subl  $12, %esp     # allineamento?
	pushl -28(%ebp)     # cinque
	pushl -24(%ebp)     # quattro
	pushl -20(%ebp)     # tre
	pushl -16(%ebp)     # due
	pushl -12(%ebp)     # uno
	call  addint5
	addl  $32, %esp      # pulisci lo stack con allineamento
```

```
	movl  %eax, -32(%ebp) # risultato in "sei"

	movl  $.LC1, %eax   # indirizzo di stringa in eax
	subl  $4, %esp
	pushl -32(%ebp)     # sei
	pushl -28(%ebp)     # cinque
	pushl -24(%ebp)     # quattro
	pushl -20(%ebp)     # tre
	pushl -16(%ebp)     # due
	pushl -12(%ebp)     # uno
	pushl %eax          # indirizzo di stringa
	call  printf
	addl  $32, %esp

	subl  $12, %esp     # allineamento?
	                     # indirizzo di
	pushl $.LC2         # stringa "addio mondo crudele"
	call  puts
	addl  $16, %esp     # pulisci lo stack con allineamento


	movl  -4(%ebp), %ecx
# restore ecx dal salvataggio di ottimizzazione

#	leave
	# quello che segue sostituisce una istruzione "leave"
		movl  %esp, %ebp
		popl  %ebp
	# fine sostituzione

	leal  -4(%ecx), %esp
# ripristina il valore non allineato di esp (e quindi
# fa sia pop che riallineamento)

	ret
	.size main, .-main
	.ident      "GCC: (Ubuntu 4.4.3-4ubuntu5) 4.4.3"
	.section    .note.GNU-stack,"",@progbits"
```

6. 19 maggio 2011

Termine del' analisi del programma precedente. Approfondimenti sulle tecniche di chiamata a subroutine e di salvataggio all' inizio e ripristino alla fine dell' ambiente da parte delle subroutine.

- o -

La ricorsività, teoria e pratica:

il programma Hanoi scritto in c, tradotto in assembly, tradotto a mano in assembly as88, ed eseguito passo a passo con il tracer t88 per vedere l' evoluzione dello stack.:

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/Esercitazioni/SoloHanoi/hanoi.c

```
#include <stdio.h>

void hanoi (int n, int da, int a, int comodo)
{
 if (n>0) {
   hanoi (n-1, da, comodo, a);
   printf ("muovi il disco %d da %d a %d \n",n, da, a);
   hanoi (n-1, comodo, a, da);
          }
 return;
}

int main()
{
     int n;
     scanf ("%d",&n);
     hanoi(n,1,2,3);
     return;
}
```

Compilato con il comando gcc -S

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/Esercitazioni/SoloHanoi/hanoi.c.s

```
     .file "hanoi.c"
     .section   .rodata
     .align 4
.LC0:
     .string    "muovi il disco %d da %d a %d \n"
     .text
.globl hanoi
     .type hanoi, @function
hanoi:
     pushl %ebp
     movl  %esp, %ebp
     subl  $8, %esp
     cmpl  $0, 8(%ebp)
     jle   .L3
     movl  8(%ebp), %eax
     subl  $1, %eax
     pushl 16(%ebp)
     pushl 20(%ebp)
     pushl 12(%ebp)
     pushl %eax
     call  hanoi
     addl  $16, %esp
     movl  $.LC0, %eax
     pushl 16(%ebp)
     pushl 12(%ebp)
     pushl 8(%ebp)
     pushl %eax
     call  printf
     addl  $16, %esp
```

```
        movl  8(%ebp), %eax
        subl  $1, %eax
        pushl 12(%ebp)
        pushl 16(%ebp)
        pushl 20(%ebp)
        pushl %eax
        call  hanoi
        addl  $16, %esp
.L3:
        leave
        ret
        .size hanoi, .-hanoi
        .section   .rodata
.LC1:
        .string    "%d"
        .text
.globl main
        .type main, @function
main:
        leal  4(%esp), %ecx
        andl  $-16, %esp
        pushl -4(%ecx)
        pushl %ebp
        movl  %esp, %ebp
        pushl %ecx
        subl  $20, %esp
        movl  $.LC1, %eax
        subl  $8, %esp
        leal  -12(%ebp), %edx
        pushl %edx
        pushl %eax
        call  __isoc99_scanf
        addl  $16, %esp
        movl  -12(%ebp), %eax
        pushl $3
        pushl $2
        pushl $1
        pushl %eax
        call  hanoi
        addl  $16, %esp
        movl  -4(%ebp), %ecx
        leave
        leal  -4(%ecx), %esp
        ret
        .size main, .-main
        .ident     "GCC: (Ubuntu 4.4.3-4ubuntu5) 4.4.3"
        .section   .note.GNU-stack,"",@progbits
```

TRADOTTO A MANO in assembler 8088:

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/Esercitazioni/SoloHanoi/hanoi88.s

```
_EXIT   = 1
_PRINTF = 127

.SECT  .TEXT

main:
        push    3   ! comodo
        push    2   ! destinazione
        push    1   ! partenza
        push    3   ! numero dischi
        call    hanoi
        add     sp, 8
        PUSH 0
        PUSH _EXIT
        SYS
hanoi:
        push    bp
        mov     bp, sp
        sub     sp, 4
        mov     ax, 4(bp)
        cmp     ax , 0
        jle         L1
        push    8(bp)
        push    10(bp)
        push    6(bp)
        mov     ax, 4(bp)
        dec     ax
        push    ax
        call    hanoi
        add     sp, 8
        push    8(bp)
        push    6(bp)
        push    4(bp)
        push    LC0
        push    _PRINTF
        SYS
        add     sp, 10
        push    6(bp)
        push    8(bp)
        push    10(bp)
        mov     ax, 4(bp)
        dec     ax
        push    ax
        call    hanoi
        add     sp, 8
L1:
        mov     sp, bp
        pop     bp
        ret

.SECT .DATA
LC0:
        .ASCIZ  "muovi il disco %d da %d a %d \n"
```

Verifica del comportamento con il simulatore.

7. 25 maggio 2011

Programmazione strutturata in assembler

Richiamo del teorema di Bohm Jacopini

Le strutture fondamentali:

Sequenza S1; S2; ... Sn. Richiamo alla strutturazione in subroutines.

If C then S1 else S2. Annidamento di If. Linearizzazione dell' annidamento con le strutture:

If C1 then S1 else if C2 then S2 else if... else if C(n-a) then S(n-1) else Sn

Case of (control): V1 S1 V2 S2 .... V(n-1) S(n-1) default Sn

Linearizzazione della struttura bidimensionale con “go to” (jump) per mantenerne le relazioni topologiche

Strutture iterative:

While C do S (while (C) {S} in C/C++)

Repeat S until C (do {} while (!C) in C/C++)

Struttura che le comprende entrambe:

While S1 gives C do S2

Come simularla in C/C++ sfruttando i side effects:

While (S1() == C) {S2}

Il For (Sinit; Stest; Smodify)

Realizzazioni in c e traduzione in assembler:

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/Esercitazioni/Strutture/strutt1.c

```
int main()
{
     int uno, due, tre;
     if (uno>0)
{
   prima();
}
     else if (uno==0)
{
   seconda();
}
     else
{
   terza();
}

   uno = quarta();


     while (whilecond()>0)
{
   whilesub(uno);
}

   uno = quinta();

       do
{
   untilsub(uno);
}
     while (uno>100);


   uno = sesta ();

     for (uno=0; uno<10;uno++)
{
   forsub (uno);
}

     return;
}
```

Compilato in assembler con il comando

```
as --32 -a $file.s -o $file.o > $file.s.list
```

diventa così:

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/Esercitazioni/Strutture/strutt1.c.s

```
        .file "strutt1.c"
        .text
.globl main
        .type main, @function
main:
        leal 4(%esp), %ecx
        andl $-16, %esp
        pushl -4(%ecx)
        pushl %ebp
        movl %esp, %ebp
        pushl %ecx
        subl $20, %esp
        call routineinizio
        cmpl $0, -12(%ebp)
        jle  .L2
        call prima
        jmp  .L3
.L2:
        cmpl $0, -12(%ebp)
        jne  .L4
        call seconda
        jmp  .L3
.L4:
        call terza
.L3:
        call quarta
        movl %eax, -12(%ebp)
        jmp  .L5
.L6:
        subl $12, %esp
        pushl -12(%ebp)
        call whilesub
        addl $16, %esp
.L5:
        call whilecond
        testl %eax, %eax
        jg   .L6
        call quinta
        movl %eax, -12(%ebp)
.L7:
        subl $12, %esp
        pushl -12(%ebp)
        call untilsub
        addl $16, %esp
        cmpl $100, -12(%ebp)
        jg   .L7
        call sesta
        movl %eax, -12(%ebp)
        movl $0, -12(%ebp)
        jmp  .L8
.L9:
        subl $12, %esp
```

```
        pushl -12(%ebp)
        call  forsub
        addl  $16, %esp
        addl  $1, -12(%ebp)
.L8:
        cmpl  $9, -12(%ebp)
        jle   .L9
        movl  -4(%ebp), %ecx
        leave
        leal  -4(%ecx), %esp
        ret
        .size main, .-main
        .ident     "GCC: (Ubuntu 4.4.3-4ubuntu5) 4.4.3"
        .section   .note.GNU-stack,"",@progbits
```

Compilato invece con il comando

```
gcc -m32 -mno-accumulate-outgoing-args -mpreferred-stack-boundary=2 -c -g
-Wa,-a,-ad   strutt1.c > strutt1.listac-as
```

Troviamo nel listato SIA il codice c CHE il codice assembler:

http://trusso.freeshell.org/Architettura2010-2011/ProgrammaEMaterialeDidattico/Esercitazioni/Strutture/strutt1.listac-as

```
   1                        .file"strutt1.c"

   9
  10
  11                      .globl main
  13                      main:
  14
  15                           .file 1 "strutt1.c"
   1:strutt1.c     ****
   2:strutt1.c     **** int main()
   3:strutt1.c     **** {
  16
  17
  18 0000 55               pushl%ebp
  19
  20
  21 0001 89E5             movl %esp, %ebp
  22
  23                      .LCFI1:
  24
  25 0003 83EC0C           subl $12, %esp
   4:strutt1.c     ****      int uno, due, tre;
   5:strutt1.c     ****
   6:strutt1.c     ****      routineinizio();
  26
  27 0006 E8FCFFFF         call routineinizio
  27      FF
   7:strutt1.c     ****
   8:strutt1.c     ****      if (uno>0)
  28
  29 000b 837DFC00         cmpl $0, -4(%ebp)
  30 000f 7E07             jle  .L2
   9:strutt1.c     **** {   prima(); }
  31
  32 0011 E8FCFFFF         call prima
  32      FF
  33 0016 EB12             jmp  .L3
  34                      .L2:
  10:strutt1.c     ****      else if (uno==0)
  35
  36 0018 837DFC00         cmpl $0, -4(%ebp)
  37 001c 7507             jne  .L4
  11:strutt1.c     **** {   seconda(); }
  38
  39 001e E8FCFFFF         call seconda
  39      FF
  40 0023 EB05             jmp  .L3
  41                      .L4:
  12:strutt1.c     ****      else
  13:strutt1.c     **** {   terza();  }
  42
  43 0025 E8FCFFFF         call terza
  43      FF
  44                      .L3:
```

```
14:strutt1.c     ****
15:strutt1.c     ****    uno = quarta();
45
46 002a E8FCFFFF         call quarta
46      FF
47 002f 8945FC           movl %eax, -4(%ebp)
16:strutt1.c     ****
17:strutt1.c     ****      while (whilecond()>0)
48
49 0032 EB0B             jmp  .L5
50                 .L6:
18:strutt1.c     ****    { whilesub(uno); }
51
52 0034 FF75FC           pushl-4(%ebp)
53
54 0037 E8FCFFFF         call whilesub
54      FF
55 003c 83C404           addl $4, %esp
56                 .L5:
17:strutt1.c     ****      while (whilecond()>0)
57
58
59 003f E8FCFFFF         call whilecond
59      FF
60 0044 85C0             testl%eax, %eax
61 0046 7FEC             jg   .L6
19:strutt1.c     ****
20:strutt1.c     ****    uno = quinta();
62
63 0048 E8FCFFFF         call quinta
63      FF
64 004d 8945FC           movl %eax, -4(%ebp)
65                 .L7:
21:strutt1.c     ****
22:strutt1.c     ****      do   { untilsub(uno); }
66
67 0050 FF75FC           pushl-4(%ebp)
68
69 0053 E8FCFFFF         call untilsub
69      FF
70 0058 83C404           addl $4, %esp
23:strutt1.c     ****      while (uno>100);
71
72 005b 837DFC64         cmpl $100, -4(%ebp)
73 005f 7FEF             jg   .L7
24:strutt1.c     ****
25:strutt1.c     ****    uno = sesta ();
74
75
76 0061 E8FCFFFF         call sesta
76      FF
77 0066 8945FC           movl %eax, -4(%ebp)
26:strutt1.c     ****
27:strutt1.c     ****      for (uno=0; uno<10;uno++)
78
79 0069 C745FC00         movl $0, -4(%ebp)
```

```
  79      000000
  80 0070 EB0F            jmp   .L8
  81                    .L9:
  28:strutt1.c     ****   {  forsub (uno);}
  82
  83 0072 FF75FC          pushl -4(%ebp)
  84
  85 0075 E8FCFFFF        call  forsub
  85      FF
  86 007a 83C404          addl  $4, %esp
  27:strutt1.c     ****     for (uno=0; uno<10;uno++)
  87
  88 007d 8345FC01        addl  $1, -4(%ebp)
  89                    .L8:
  90 0081 837DFC09        cmpl  $9, -4(%ebp)
  91 0085 7EEB            jle   .L9
  29:strutt1.c     ****
  30:strutt1.c     ****     return;
  31:strutt1.c     **** }
  92
  93 0087 C9              leave
  94 0088 C3              ret
  95
  96
  98  hh


DEFINED SYMBOLS
                         *ABS*:0000000000000000 strutt1.c
     /tmp/cccE1Ajb.s:13     .text:0000000000000000 main

UNDEFINED SYMBOLS
routineinizio
prima
seconda
terza
quarta
whilesub
whilecond
quinta
untilsub
sesta
forsub
```

Esame della traduzione.

Riepilogo generale

Tommaso Russo

Nato nel 1947, fisico teorico per vocazione, fulminato lungo questa strada dall' informatica nel 1970, non se ne è più staccato. Ha operato principalmente nei settori dei sistemi operativi, dei linguaggi, del supercalcolo, degli algoritmi e delle reti, alle dipendenze di un produttore "storico" (L' Univac), dell' Università di Trieste (dove ha prodotto gran parte delle sue pubblicazioni), dell' Area Science Park (dove ha progettato e realizzato la rete interna e contribuito allo sviluppo della rete GARR).

Attualment progetta sistemi telematici basati su terminali radiomobili e
insegna all' Università e all' Area ogniqualvolta ne ha l' occasione.

Lo trovate a uno di questi indirizzi:

russo@univ.trieste.it
tommasoalbertorusso@virgilio.it
trusso@tin.it