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 ed esposizione del piano del corso (vedi programma – manifesto del corso)

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

## 6. 20 aprile 2011

Validazione W3C HTML
Uso intelligente di motori di ricerca

## 7. 4 maggio 2011

# 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 non accetta 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://enteos2.area.trieste.it/russo/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://enteos2.area.trieste.it/russo/Architettura2007-2008/ProgrammaEMaterialeDidattico/Esercitazioni/CompileLink/addintmain.c | http://enteos2.area.trieste.it/russo/Architettura2007-2008/ProgrammaEMaterialeDidattico/Esercitazioni/CompileLink/addintsubs.c |
|---|---|
| `#include <stdio.h>`<br><br>`int main ()`<br>`  {`<br>`  int uno;`<br>`  int due;`<br>`  int tre;`<br><br>`  uno = 257;`<br>`  due = 64;`<br>`  tre = addint (uno,due);`<br>`  printf ("%d + %d = %d \n" , uno, due, tre);`<br>`  tre = multint (uno,due);`<br>`  printf ("%d * %d = %d \n" , uno, due, tre);`<br>`  }` | `int addint(int a, int b)`<br>`{`<br>`int c;`<br>`c = a+b;`<br>`return c;`<br>`}`<br><br>`int multint(int a, int b)`<br>`{`<br>`int c;`<br>`c = a * b;`<br>`return c;`<br>`}` |

Esecuzione della catena passo 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  addintsubs.c -S -masm=intel -o  addintsubs.s
```

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

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
```

assembling dei risultati con il comando

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

Confronto dei risultati:

***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


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
```

***Esempi di listato assembler***

```
   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
   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)
```

***Esempi di listato assembler***

```
  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)
  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 Penttium (Tanenbaum cap. 5)

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

Esame di un programma assemler:

- direttive al compilatore:
  - Sezioni `.SECT .TEXT, .DATA, .BSS`
  - Nomi simbolici `(es. _WRITE=4, _STDOUT = 1)`
- Labels
  - Necessità del doppio passaggio Assembler per la creazione della Symbol Table

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

http://enteos2.area.trieste.it/russo/Architettura2007-2008/ProgrammaEMaterialeDidattico/consultazione/ManualiIntel/

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

## 3. 1 aprile 2008

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)**

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://enteos2.area.trieste.it/russo/Architettura2007-2008/ProgrammaEMaterialeDidattico/consultazione/ManualiIntel/INTEL-2A-253666.pdf

http://enteos2.area.trieste.it/russo/Architettura2007-2008/ProgrammaEMaterialeDidattico/consultazione/ManualiIntel/INTEL-2B-253666.pdf

## 4. 4 aprile 2008

Completamento analisi sistematica delle principali istruzioni 8088

**(Tanenbaum cap. C.4, figura C.4)**

Uso di `t88` (Da CD Tanenbaum; descritto in **Tanenbaum C.6**)

ESERCITAZIONE: Modifica, rissemblaggio tramite `as88` di un programma "universale" (ottenuto modificando Hello Word) e sua esecuzione step by step tramite `t88`; analisi dei risultati: materiale:

Modifica tramite pico o emacs di

**`/home/trusso/8088_tra/linux/examples/HlloWord.s`**

per inserirvi le istruzioni dopo la linea commento "`!istruzioni aggiunte per testarne il funzionamento`" e modifica del messaggio finale

```
! Simple "hello world" program

        _EXIT   = 1             !  1
        _WRITE  = 4             !  2
        _STDOUT = 1             !  3
.SECT .TEXT                     !  4
start:                          !  5

! istruzioni aggiunte per testare il funzionamento
   MOV AX,de
   MOV AX,hw
   MOV AX,(hw)
! fine istruzioni aggiunte

        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  "addio mondo crudele \n"! 19
de:     .BYTE   0               ! 20
.SECT .BSS
```

Copia della directory **`/home/trusso//8088_tra/linux/examples/`** in una directory privata dello studente; CD nella directory privata dello studente; comandi

```
./as88 HlloWord.s
./t88 HlloWord.s
```

esecuzione step by step con analisi dei registri dopo ogni istruzione.

## 5. 11 aprile 2008

**Programmazione strutturata in Assembler: chiamata a subroutines**

Verifica dell' esempio GenReg con il comando

```
./t88 GenReg.s
```

esecuzione step by step con analisi dei registri dopo ogni istruzione.

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

http://enteos2.area.trieste.it/russo/Architettura2007-2008/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://enteos2.area.trieste.it/russo/Architettura2007-2008/ProgrammaEMaterialeDidattico/Esercitazioni/ChiamateSubInC/TestCallMod.s**

```
        .intel_syntax
        .text
.globl fanulla
        .type   fanulla, @function
fanulla:
        push    %ebp
        mov     %ebp, %esp
        # quello che segue sostituisce una istruzione "leave"
                mov     %esp, %ebp
                pop     %ebp
        # fine sostituzione
        ret
        .size   fanulla, .-fanulla
.globl dammizero
        .type   dammizero, @function
dammizero:
        push    %ebp
        mov     %ebp, %esp
        mov     %eax, 0
        # quello che segue sostituisce una istruzione "leave"
                mov     %esp, %ebp
                pop     %ebp
        # fine sostituzione
        ret
        .size   dammizero, .-dammizero
.globl addint2
        .type   addint2, @function
addint2:
        push    %ebp
        mov     %ebp, %esp
        mov     %eax, DWORD PTR [%ebp+12] # come dire: %ebp + 3*4, ossia
3*"Architetture/8"
                                        # cioe' PENULTIMO argomento messo in push
                                        # (ossia SECONDO nel C)
        add     %eax, DWORD PTR [%ebp+8]  # 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)

        # a questo punto eax contiene gia' il risultato finale, non resta niente
da fare

        # quello che segue sostituisce una istruzione "leave"
                mov     %esp, %ebp
                pop     %ebp
        # fine sostituzione
        ret
        .size   addint2, .-addint2
.globl addint3
        .type   addint3, @function
addint3:
        push    %ebp
        mov     %ebp, %esp
        mov     %eax, DWORD PTR [%ebp+12]    # SECONDO argomento della call
        add     %eax, DWORD PTR [%ebp+8]     # PRIMO argomento della call
        add     %eax, DWORD PTR [%ebp+16]    # TERZO argomento della call
# a questo punto eax contiene gia' il risultato finale, non resta niente da fare

        # quello che segue sostituisce una istruzione "leave"
                mov     %esp, %ebp
                pop     %ebp
        # fine sostituzione
        ret
        .size   addint3, .-addint3
.globl addint4
        .type   addint4, @function
addint4:
        push    %ebp
        mov     %ebp, %esp
        mov     %eax, DWORD PTR [%ebp+12]    # SECONDO argomento della call
        add     %eax, DWORD PTR [%ebp+8]     # PRIMO argomento della call
        add     %eax, DWORD PTR [%ebp+16]    # TERZO argomento della call
        add     %eax, DWORD PTR [%ebp+20]    # QUARTO argomento della call
        # quello che segue sostituisce una istruzione "leave"
                mov     %esp, %ebp
                pop     %ebp
        # fine sostituzione
        ret
        .size   addint4, .-addint4
        .section        .rodata
        .align 4
.LC0:
        .string "somma: %d somma2: %d somma3: %d\n"
        .text
.globl addint5
        .type   addint5, @function
addint5:
        push    %ebp
        mov     %ebp, %esp
        # sub   %esp, 24    # riserva spazio per SEI variabili (anche se ne
abbiamo dichiarate tre);
                        # problemi di allineamento?
                        # proviamo a riservare spazio solo per tre:
        sub     %esp, 12

        mov     %eax, DWORD PTR [%ebp+12]    # SECONDO argomento della call (b)
        add     %eax, DWORD PTR [%ebp+8]     # PRIMO argomento della call (a)
        add     %eax, DWORD PTR [%ebp+16]    # TERZO argomento della call (c)
```

```
        add     %eax, DWORD PTR [%ebp+20]    # QUARTO argomento della call (d)
        add     %eax, DWORD PTR [%ebp+24]    # QUINTO argomento della call (e)

# Qui la cosa si fa interessante: DOVE viene messo il risultato?

        mov     DWORD PTR [%ebp-4], %eax     # [%ebp-4] corrisponde a "somma"


        sub     %esp, 8
        push    DWORD PTR [%ebp+8]       # a
        push    DWORD PTR [%ebp-4]       # somma
        call    addint2
        add     %esp, 16
        mov     DWORD PTR [%ebp-8], %eax      # [%ebp-8] corrisponde a "somma2"

        sub     %esp, 4
        push    DWORD PTR [%ebp+12]       # b
        push    DWORD PTR [%ebp+8]        # a
        push    DWORD PTR [%ebp-4]        # somma
        call    addint3
        add     %esp, 16
        mov     DWORD PTR [%ebp-12], %eax     # [%ebp-12] corrisponde a "somma3"
                        # infatti: chiamata a printf per stamparla
                        # (stringa, somma, somma2 e somma3 in ordine inverso!)

        push    DWORD PTR [%ebp-12]
        push    DWORD PTR [%ebp-8]
        push    DWORD PTR [%ebp-4]
        push    OFFSET FLAT:.LC0
        call    printf
        add     %esp, 16
        # fine chiamata a printf

        mov     %eax, DWORD PTR [%ebp-4]  # return somma viene fatto qui
        leave
        ret
        .size   addint5, .-addint5
        .section        .rodata
        .align 4
.LC1:
        .string "uno: %d due: %d tre: %d quattro: %d cinque: %d sei: %d \n"
.LC2:
        .string "addio mondo crudele\n"
        .text
.globl main
        .type   main, @function
main:
        push    %ebp
        mov     %ebp, %esp
        sub     %esp, 24     # riserva spazio per SEI variabili
        and     %esp, -16    # e allinea al blocco di 16 byte (ottimizzazione)
        mov     %eax, 0
        sub     %esp, %eax
        mov     DWORD PTR [%ebp-4], 257     # variabile "uno"
        call    dammizero
        mov     DWORD PTR [%ebp-8], %eax    # variabile "due"

        sub     %esp, 8
        push    DWORD PTR [%ebp-8]
        push    DWORD PTR [%ebp-4]
        call    addint2
        add     %esp, 16
```

```
mov     DWORD PTR [%ebp-12], %eax

sub     %esp, 4  # sempre per allineamento
push    DWORD PTR [%ebp-12]    # variabile "tre"
push    DWORD PTR [%ebp-8]     # variabile "due"
push    DWORD PTR [%ebp-4]     # variabile "uno"
call    addint3
add     %esp, 16
mov     DWORD PTR [%ebp-16], %eax


push    DWORD PTR [%ebp-16]   # variabile "quattro"
push    DWORD PTR [%ebp-12]
push    DWORD PTR [%ebp-8]
push    DWORD PTR [%ebp-4]
call    addint4
add     %esp, 16    #(nota: qui non e' servito l' allineamento)
mov     DWORD PTR [%ebp-20], %eax

sub     %esp, 12 # sempre per allineamento
push    DWORD PTR [%ebp-20]   # variabile "cinque"
push    DWORD PTR [%ebp-16]
push    DWORD PTR [%ebp-12]
push    DWORD PTR [%ebp-8]
push    DWORD PTR [%ebp-4]
call    addint5
add     %esp, 32
mov     DWORD PTR [%ebp-24], %eax
sub     %esp, 4      # sempre per allineamento
push    DWORD PTR [%ebp-24]
push    DWORD PTR [%ebp-20]
push    DWORD PTR [%ebp-16]
push    DWORD PTR [%ebp-12]
push    DWORD PTR [%ebp-8]
push    DWORD PTR [%ebp-4]
push    OFFSET FLAT:.LC1
call    printf
add     %esp, 32

sub     %esp, 12            # sempre per allineamento
push    OFFSET FLAT:.LC2
call    printf
add     %esp, 16
# quello che segue sostituisce una istruzione "leave"
        mov     %esp, %ebp
        pop     %ebp
# fine sostituzione
ret
.size   main, .-main
.section        .note.GNU-stack,"",@progbits
.ident  "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

## 6. 18 aprile 2008

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://enteos2.area.trieste.it/russo/Architettura2007-2008/ProgrammaEMaterialeDidattico/Esercitazioni/hanoi/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;
}
```

http://enteos2.area.trieste.it/russo/Architettura2007-2008/ProgrammaEMaterialeDidattico/Esercitazioni/hanoi/hanoi.s

```
          .file   "hanoi.c"
        .intel_syntax
        .section        .rodata
        .align 4
.LC0:
        .string "muovi il disco %d da %d a %d \n"
        .text
.globl hanoi
        .type   hanoi, @function
hanoi:
        push    %ebp
        mov     %ebp, %esp
        sub     %esp, 8
        cmp     DWORD PTR [%ebp+8], 0
        jle     .L1
        push    DWORD PTR [%ebp+16]
```

```
        push    DWORD PTR [%ebp+20]
        push    DWORD PTR [%ebp+12]
        mov     %eax, DWORD PTR [%ebp+8]
        dec     %eax
        push    %eax
        call    hanoi
        add     %esp, 16
        push    DWORD PTR [%ebp+16]
        push    DWORD PTR [%ebp+12]
        push    DWORD PTR [%ebp+8]
        push    OFFSET FLAT:.LC0
        call    printf
        add     %esp, 16
        push    DWORD PTR [%ebp+12]
        push    DWORD PTR [%ebp+16]
        push    DWORD PTR [%ebp+20]
        mov     %eax, DWORD PTR [%ebp+8]
        dec     %eax
        push    %eax
        call    hanoi
        add     %esp, 16
.L1:
        leave
        ret
        .size   hanoi, .-hanoi
        .section        .rodata
.LC1:
        .string "%d"
        .text
.globl main
        .type   main, @function
main:
        push    %ebp
        mov     %ebp, %esp
        sub     %esp, 8
        and     %esp, -16
        mov     %eax, 0
        sub     %esp, %eax
        sub     %esp, 8
        lea     %eax, [%ebp-4]
        push    %eax
        push    OFFSET FLAT:.LC1
        call    scanf
        add     %esp, 16
        push    3
        push    2
        push    1
        push    DWORD PTR [%ebp-4]
        call    hanoi
        add     %esp, 16
        leave
        ret
        .size   main, .-main
        .section        .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

http://enteos2.area.trieste.it/russo/Architettura2007-2008/ProgrammaEMaterialeDidattico/Esercitazioni/hanoi/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"
```

## 1. 6 maggio 2008

**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://enteos2.area.trieste.it/russo/Architettura2007-2008/ProgrammaEMaterialeDidattico/Esercitazioni/hanoi/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;
}
```

http://enteos2.area.trieste.it/russo/Architettura2007-2008/ProgrammaEMaterialeDidattico/Esercitazioni/hanoi/strutt1.s

```
        .file   "strutt1.c"
        .intel_syntax
        .text
.globl main
        .type   main, @function
main:
        push    %ebp
        mov     %ebp, %esp
        sub     %esp, 24
        and     %esp, -16
        mov     %eax, 0
        sub     %esp, %eax
```

```
        cmp     DWORD PTR [%ebp-4], 0
        jle     .L2
        call    prima
        jmp     .L3
.L2:
        cmp     DWORD PTR [%ebp-4], 0
        jne     .L4
        call    seconda
        jmp     .L3
.L4:
        call    terza
.L3:
        call    quarta
        mov     DWORD PTR [%ebp-4], %eax
.L6:
        call    whilecond
        test    %eax, %eax
        jg      .L8
        jmp     .L7
.L8:
        sub     %esp, 12
        push    DWORD PTR [%ebp-4]
        call    whilesub
        add     %esp, 16
        jmp     .L6
.L7:
        call    quinta
        mov     DWORD PTR [%ebp-4], %eax
.L9:
        sub     %esp, 12
        push    DWORD PTR [%ebp-4]
        call    untilsub
        add     %esp, 16
        cmp     DWORD PTR [%ebp-4], 100
        jg      .L9
        call    sesta
        mov     DWORD PTR [%ebp-4], %eax
        mov     DWORD PTR [%ebp-4], 0
.L13:
        cmp     DWORD PTR [%ebp-4], 9
        jle     .L16
        jmp     .L14
.L16:
        sub     %esp, 12
        push    DWORD PTR [%ebp-4]
        call    forsub
        add     %esp, 16
        lea     %eax, [%ebp-4]
        inc     DWORD PTR [%eax]
        jmp     .L13
.L14:
        leave
        ret
        .size   main, .-main
        .section        .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

## 8. 16 maggio 2008

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