
Corso di Architettura dei Sistemi a Microprocessore

Instruction Set Architecture



Luigi Coppolino

Contact info

Prof. Luigi Coppolino
luigi.coppolino@uniparthenope.it

Università degli Studi di Napoli "Parthenope"
Dipartimento di Ingegneria

Centro Direzionale di Napoli, Isola C4
V Piano lato SUD - Stanza n. 512

Tel: +39-081-5476702
Fax: +39-081-5476777



Roadmap



The Fault and Intrusion Tolerant NETworked SystemS (FITNESS) Research Group
<http://www.fitnesslab.eu/>



Sources

- Textbook (chapter 2)
- Manuale Freescale
(http://www.freescale.com/files/archives/doc/ref_manual/M68000PRM.pdf)(http://www.freescale.com/files/dsp/doc/ref_manual/CFPRM.pdf)
- Manuale ARM
(http://infocenter.arm.com/help/topic/com.arm.doc.dui0204j/DUI0204J_rvct_assembler_guide.pdf)
- Quick Guides:
 - ARM:
http://infocenter.arm.com/help/topic/com.arm.doc.qrc0001l/QRC0001_UAL.pdf
 - Coldfire (m68000):
<http://home.anadolu.edu.tr/~sgorgulu/micro2/2008/68KISx1.pdf>



Livello Software

Software

User Level: Application Programs

High Level Languages

Assembly Language/Machine Code

Microprogrammed/Hardwired
Control

Functional Units (Memory, ALU, etc.)

Logic Gates

Transistors and Wires

User Level: Application Programs

Problem-oriented Languages

Assembly Language

Operating System

Conventional Machine

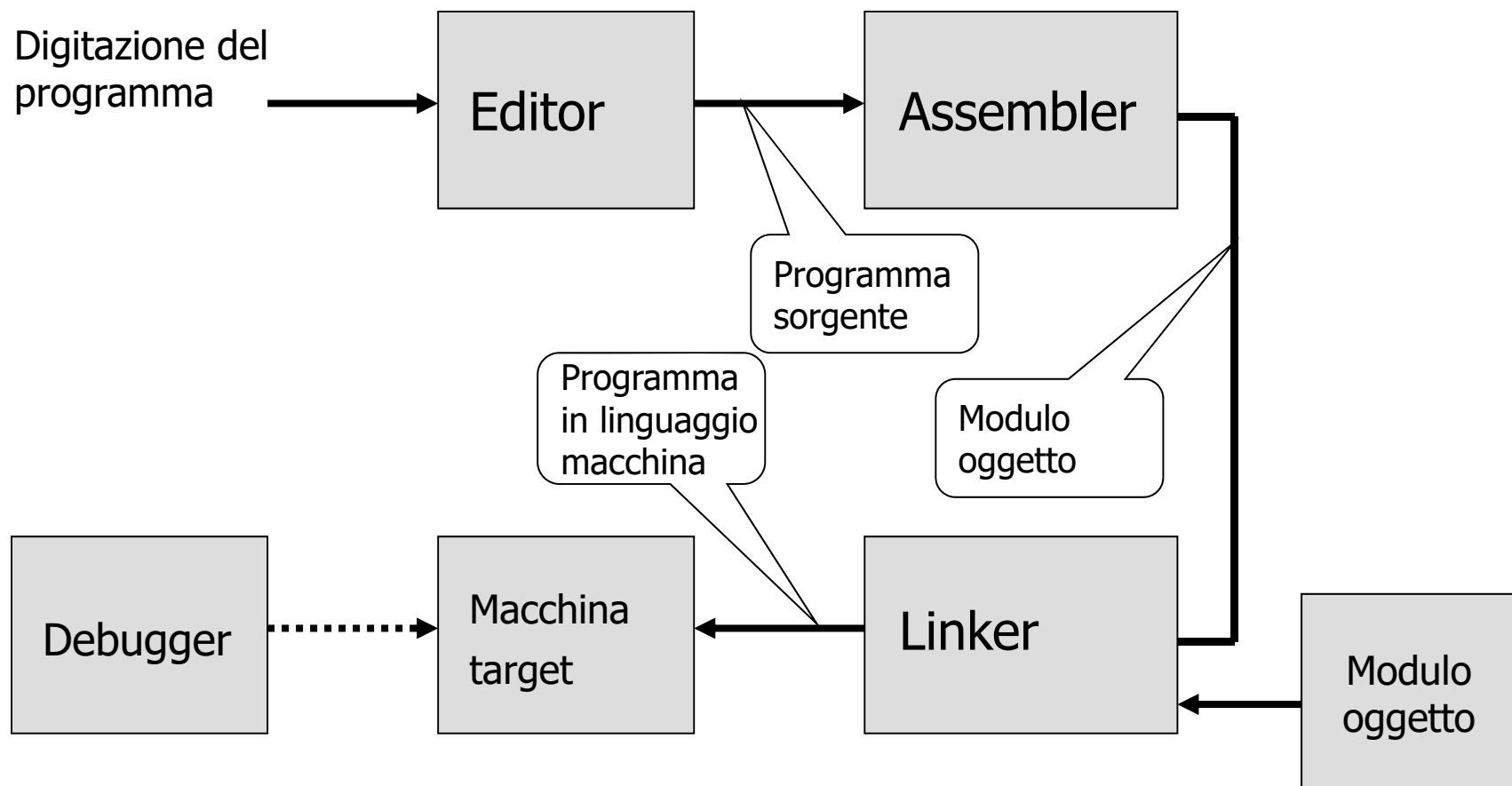
Microprogramming Level

Functional Units (Memory, ALU, etc.)

Logic Gates

Transistors and Wires

Ciclo di sviluppo



Assembly

- È funzionalmente equivalente al linguaggio macchina, ma usa “nomi” più intuitivi (mnemonics)
- Definisce l’Instruction Set Architecture (ISA) della macchina
- Un compilatore traduce un linguaggio di alto livello, che è indipendente dall’architettura, in linguaggio assembly, che è dipendente dall’architettura
- Un assembler traduce programmi in linguaggio assembly in codice binario eseguibile
- Nel caso di linguaggi compilati (es. C) il codice binario viene eseguito direttamente dalla macchina target
- Nel caso di linguaggi interpretati (es. Java) il bytecode viene interpretato dalla Java Virtual Machine, che è al livello Assembly language

Esempio – Assembly X86 a 32 bit

```
DES_std_crypt:
    movl 4(%esp),%edx
    pushl %ebx
    movl DES_count,%ecx
    xorl %ebx,%ebx
    movq (%edx),K1
    movq 32(%edx),K2
    movq K1,tmp1
    movq 8(%edx),K3
    movq 16(%edx),K4
    DES_copy(24, 40)
    ...
    DES_copy(112, 120)
    movq DES_IV,R
    xorl %edx,%edx
    movq DES_IV+8,L
DES_loop:
    ...
```

ARM - RISC

- Istruzioni: semplici ed efficienti
 - Tutte le istruzioni sono lunghe 32 bit -> semplifica fase di fetch
 - Codifica regolare -> semplifica fase di decodifica
 - Quasi tutte le istruzioni aritmetiche a tre operandi
 - Formato:

OP Rd, R1, R2 [o immediato]

- Alto numero di registri generali
- Frequenze di clock elevate
- Compilazione e Debugging più complesse

Coldfire – CISC

- Istruzioni: complesse e a lunghezza variabile
- Una, due o tre parole consecutive
 - *OP-code* nella prima parola, specifica operazione
 - Fornisce anche informazioni su come recuperare gli operandi
 - Eventualmente una o due *extension words*
- Quasi tutte le operazioni, aritmetiche e di trasferimento dati, hanno due operandi :
OP src, dst
 - La dimensione dell'operando è specificata dal suffisso .B, .W, .L

Codice Assembly

```
MOVE.B  N,D0
LEA     V,A0

contaPari
    MOVE.B  D2,-(SP)
    MOVE.B  D1,-(SP)
    MOVE.B  #0,D1    ; contatore
loop
    MOVE.B  (A0)+,D2
    AND.B   #%00000001,D2
    BNE disp
    ADDQ.B  #1,D1
disp ADD.B  #-1,D0
    BGT     loop

    ORG $2000
N    DC.B   5
V    DC.B   3,6,1,2,8
```

Formato del Codice Assembly

LABEL **OPCODE** *OP1, OP2, OP3* ; comment

- Una linea di codice sorgente Assembly è costituita da quattro campi:
 - **LABEL**: Stringa alfanumerica, definisce un nome simbolico per il corrispondente indirizzo
 - Utilizzabile per definire una variabile
 - **OPCODE**: Codice mnemonico o pseudo-operatore, determina la generazione di un'istruzione in linguaggio macchina o la modifica del valore corrente del Program Location Counter
 - **OPERANDS**: Oggetti dell'azione specificata dall'OPCODE, variano a seconda dell'OPCODE
 - **COMMENTS**: Testo arbitrario inserito dal programmatore

Codice Assembly

```

00001000 1039 00002000      MOVE.B  N,D0
00001006 41F9 00002001      LEA     V,A0
...
00001016
00001016          contaPari
00001016 1F02              MOVE.B  D2,-(SP)
00001018 1F01              MOVE.B  D1,-(SP)
0000101A 123C 0000          MOVE.B  #0,D1    ; contatore
0000101E          loop
0000101E 1418              MOVE.B  (A0)+,D2
00001020 C43C 0001          AND.B   #%00000001,D2
00001024 6600 0004          BNE disp
00001028 5201              ADDQ.B  #1,D1
0000102A 0600 00FF          disp ADD.B  #-1,D0
0000102E 6EEE              BGT     loop
...
00002000          ORG $2000
00002000= 05              N      DC.B   5
00002001= 03 06 01 02 08  V      DC.B   3,6,1,2,8

```

SYMBOL TABLE INFORMATION	
Symbol-name	Value

CONTAPARI	1016
DISP	102A
LOOP	101E
N	2000
START	1000
V	2001



Livello ISA

Program Location Counter (PLC)

- E' una variabile interna dell'assemblatore
- Punta alla locazione di memoria in cui andrà caricata l'istruzione assemblata
- Viene inizializzato dallo pseudo-operatore "origin" (ORG)
- Durante il processo di assemblaggio, il suo valore è aggiornato sia in funzione degli operatori, sia in funzione degli pseudo-operatori
- In un programma è possibile fare riferimento al suo valore corrente, tipicamente mediante il simbolo "*"

0000102E	6EEE		BGT	loop
...				
00002000			ORG	\$2000
00002000=	05	N	DC.B	5
00002001=	03 06 01 02 08	V	DC.B	3, 6, 1, 2, 8

Convenzioni

- Gli spazi bianchi tra i diversi campi fungono esclusivamente da separatori (vengono ignorati dall'assemblatore)
- Una linea che inizi con un asterisco ("*/;') è una linea di commento
- Nelle espressioni assembly, gli argomenti di tipo numerico si intendono espressi
 - In notazione decimale, se non diversamente specificato
 - In notazione esadecimale, se preceduti dal simbolo '\$' o '0x'
 - In notazione binaria (0,1) se preceduti da %

Instruction Set Architecture

- E' l'interfaccia tra il programmatore ed il processore
- Definisce l'insieme di istruzioni eseguibili dal processore
- Caratterizzata da:
 - Elenco istruzioni
 - Formato e Codifica binaria delle istruzioni
 - Modi di indirizzamento

Instruction format

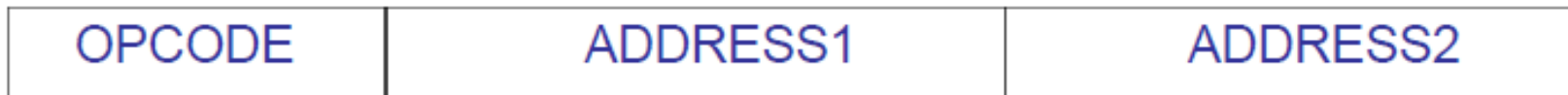
Istruzioni a 0 operandi



Istruzioni a 1 operando



Istruzioni a 2 operandi



Istruzioni a 3 operandi



Le istruzioni possono essere tutte della stessa lunghezza in bit (codifica a lunghezza fissa) oppure possono essere di lunghezze differenti (codifica a lunghezza variabile)

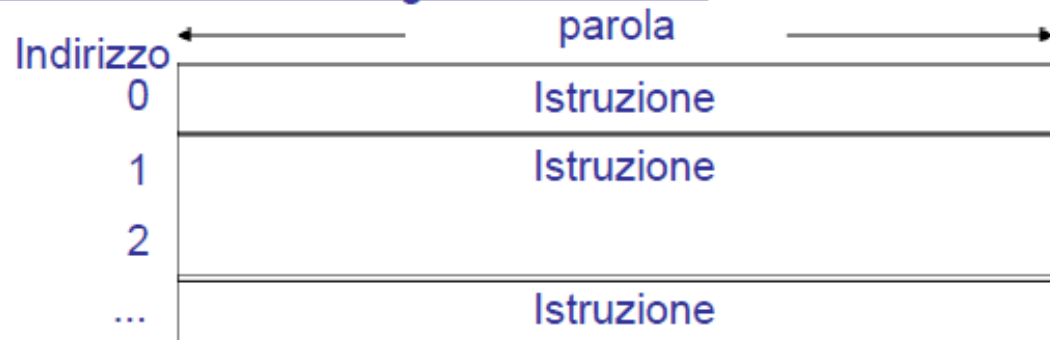
Instruction format

- Sono possibili diverse relazioni tra la lunghezza dell'istruzione e la lunghezza della parola del processore

Codifica delle istruzioni a lunghezza fissa



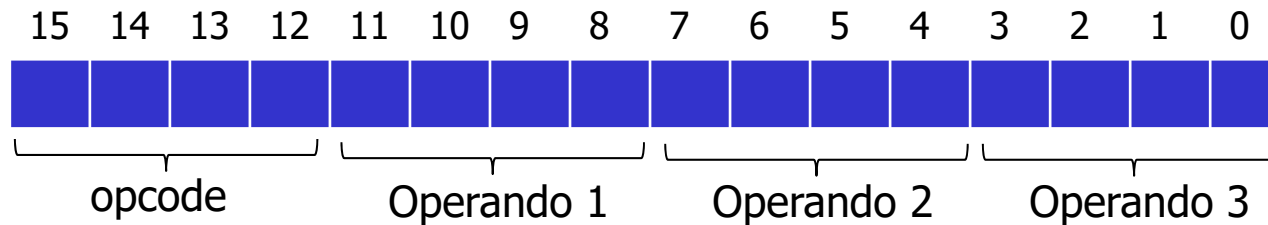
Codifica delle istruzioni a lunghezza variabile



Considerazioni

- Lunghezze minori riducono la banda necessaria al fetch delle istruzioni
- La prima parola porta con se l'opcode e le informazioni sui modi di indirizzamento necessarie a comprendere se ci sono extension word
 - Codifica a lunghezza variabile rende più complessa la fase di fetch che necessita di una parziale decodifica

Espansione dell'opcode



Un campo opcode a n bit $\Rightarrow 2^n$ diverse istruzioni

Espansione dell'opcode: riservare una configurazione di n bit per indicare che ulteriori bit sono riservati per l'opcode

Es.

Istruzioni a 3 operandi usano 4 bit di opcode ma se opcode = 0000 l'istruzione usa 8 bit di opcode e ha due operandi ...

Iterando: 12 bit di opcode e 1 operando e 16 bit opcode 0 operandi

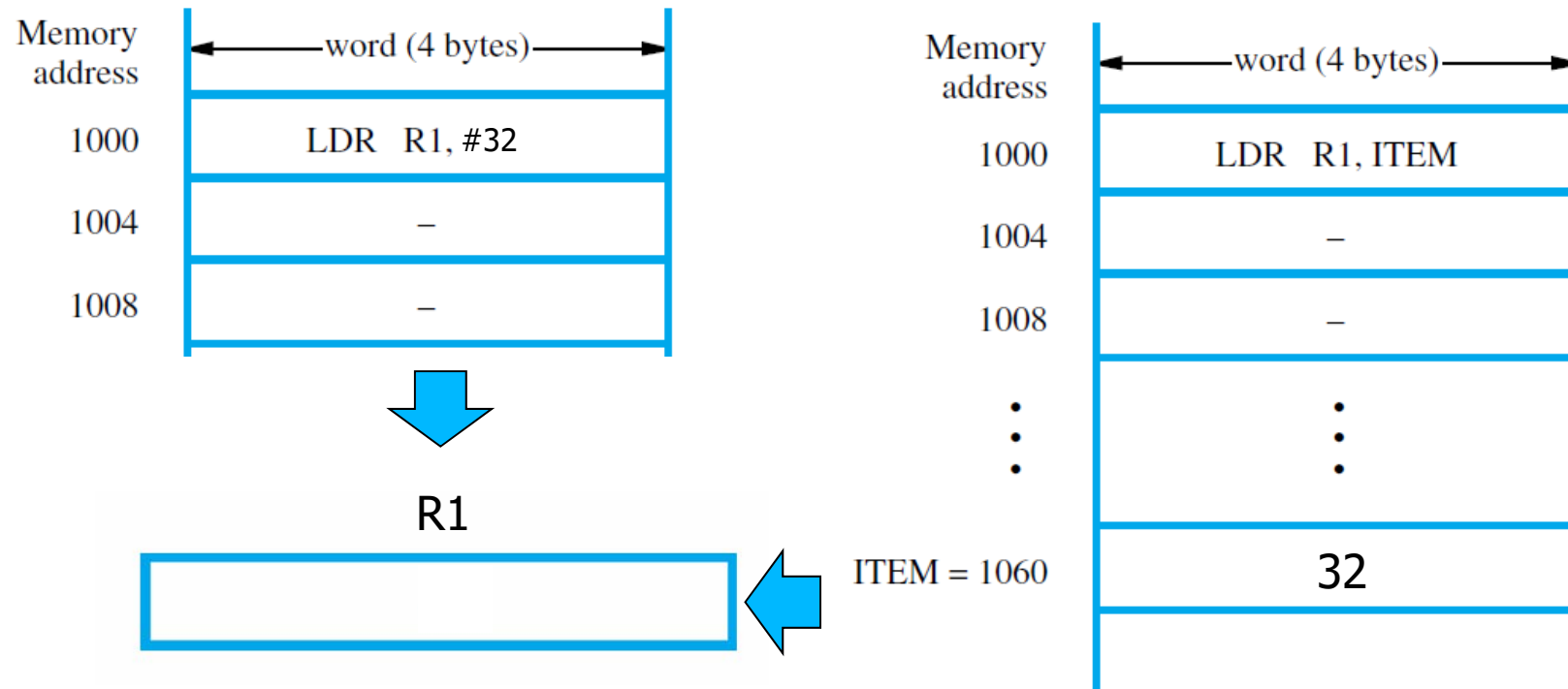


Modi di Indirizzamento

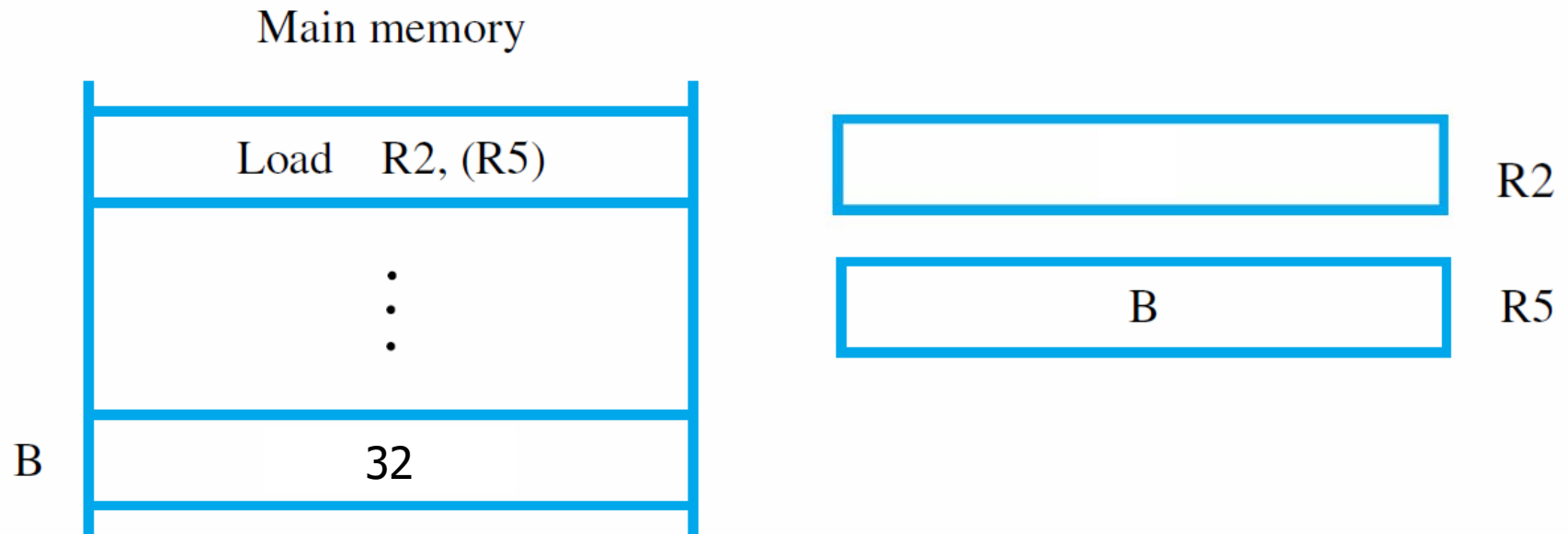
In breve

- **Immediato:** l'operando è espresso come costante nell'istruzione
- **Assoluto:** l'istruzione contiene l'indirizzo di memoria (Effective Address) dell'operando, eventualmente espresso come etichetta
- **Diretto a registro:** l'istruzione riporta il registro nel quale è contenuto l'operando
- **Indiretto a registro:** l'istruzione riporta il registro nel quale è contenuto l'indirizzo di memoria del registro
- **Indicizzato:** l'indirizzo dell'operando è dato dalla somma tra il contenuto di un registro e una costante riportati nell'istruzione
- **Con Base ed Indice:**

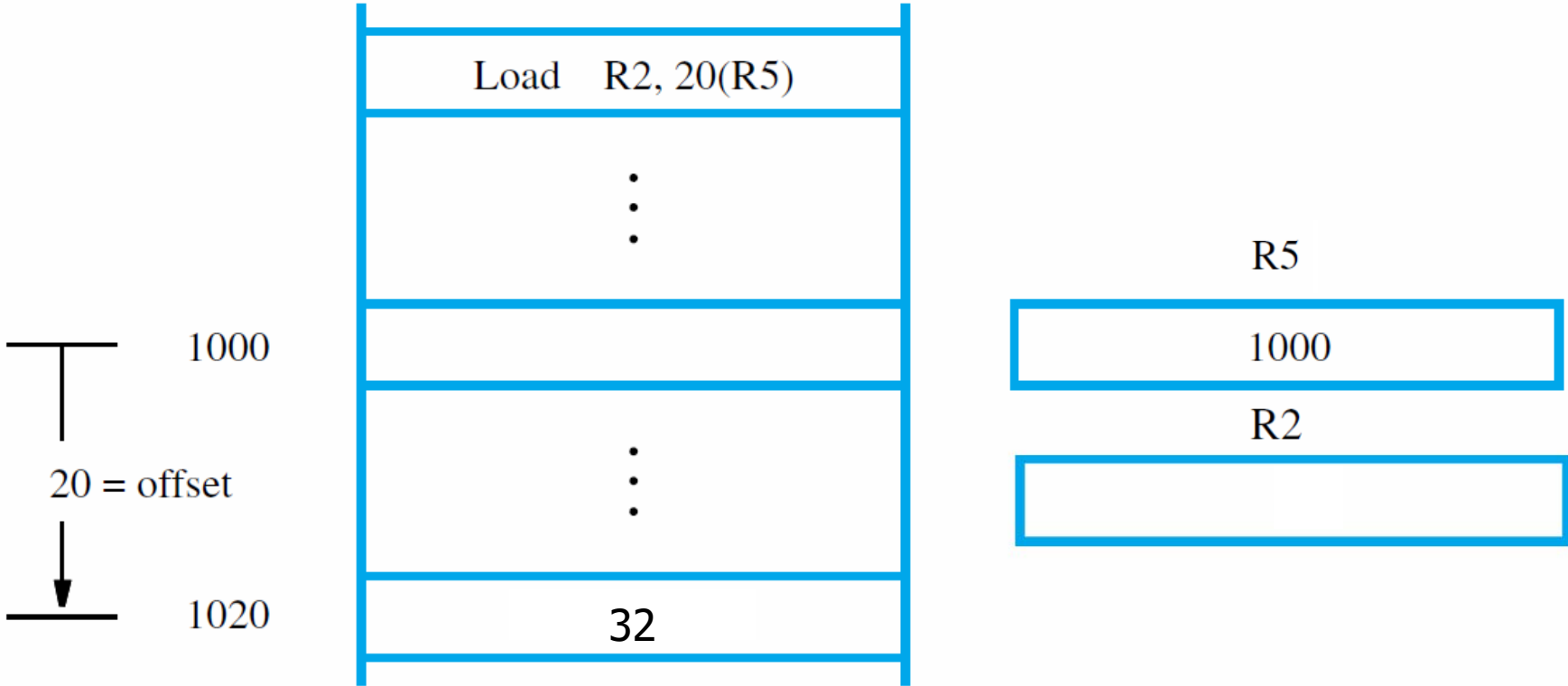
Immediato Vs Assoluto



Diretto a registro Vs Indiretto a registro

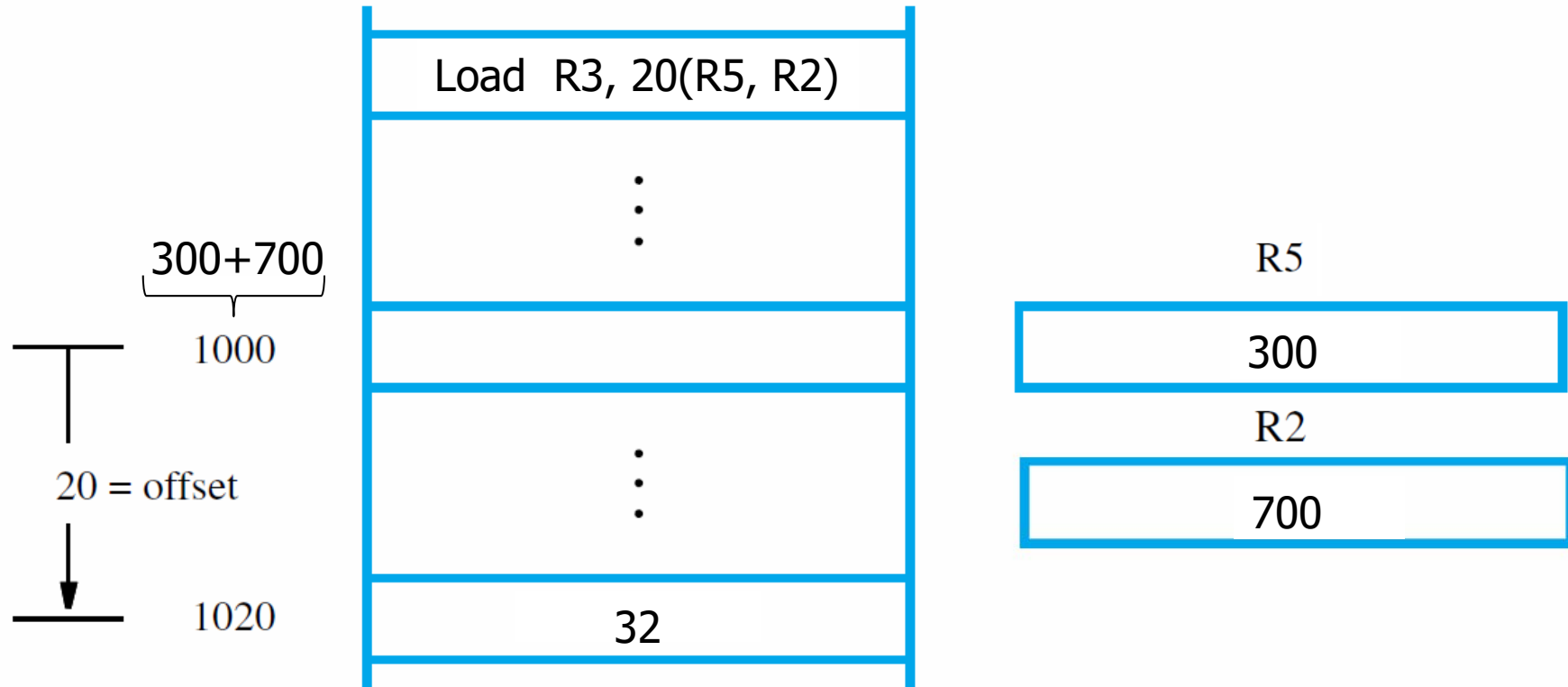


Indicizzato



(a) Offset is given as a constant

Base ed Indice



(a) Offset is given as a constant

Architetture RISC vs CISC

➤ CISC: coldfire/x86 (?)

- Istruzioni a lunghezza variabile
- Architettura con operandi in memoria
- ISA che include istruzioni complesse che richiedono molti cicli di clock
- Tendenzialmente molti modi di indirizzamento

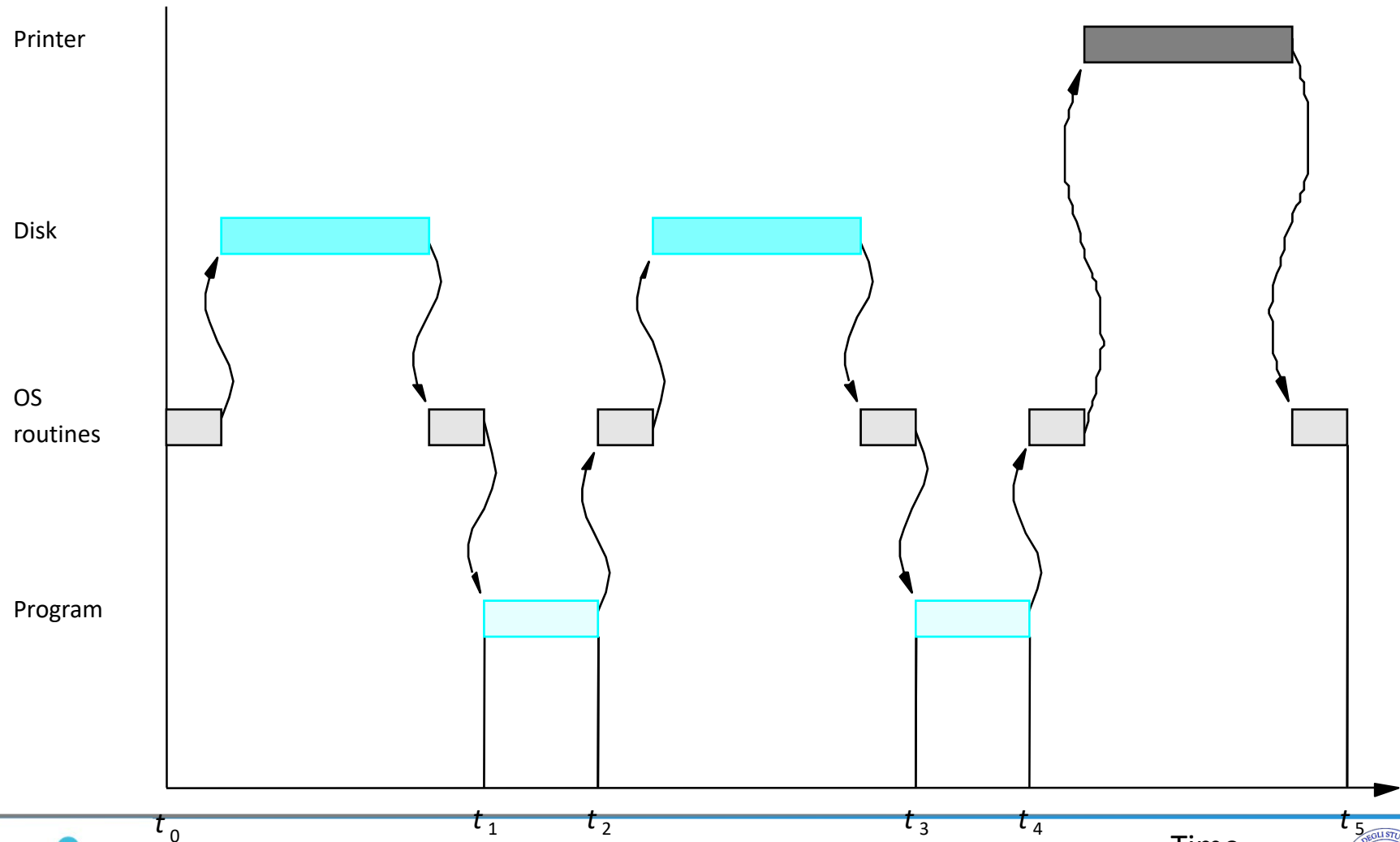
➤ RISC: ARM

- Istruzioni a lunghezza fissa
- Architettura a registri generali di tipo load-store
- ISA con istruzioni semplici
- Pochi modi di indirizzamento
- Molti registri

Performance

- Elapsed Time
- Processor Time

Sistema Monoprogrammato



The Fault and Intrusion Tolerant NETworked SystemS (FITNESS) Research Group Time

<http://www.fitnesslab.eu/>



Basic Performance Equation

$$T = (N * S) / R$$

- T = Tempo necessario ad eseguire il programma
- N = Numero di istruzioni eseguite
- S = Numero medio di cicli di clock per eseguire una istruzione
- R = clock rate

Pipelining and Superscalar Operation

- Pipelining:
 - Tecnica consistente nel sovrapporre (**overlapping**) l'esecuzione di istruzioni successive
 - Nel **caso ideale**, e cioè quando tutte le istruzioni sono sovrapposte al massimo brado possibile, l'esecuzione procede con un rate di esecuzione di **una istruzione per ciclo di clock**
- Superscalar Operation:
 - Più istruzioni sono eseguite completamente in parallelo
 - Richiede la presenza di più pipeline
 - Consente un maggior livello di concorrenza

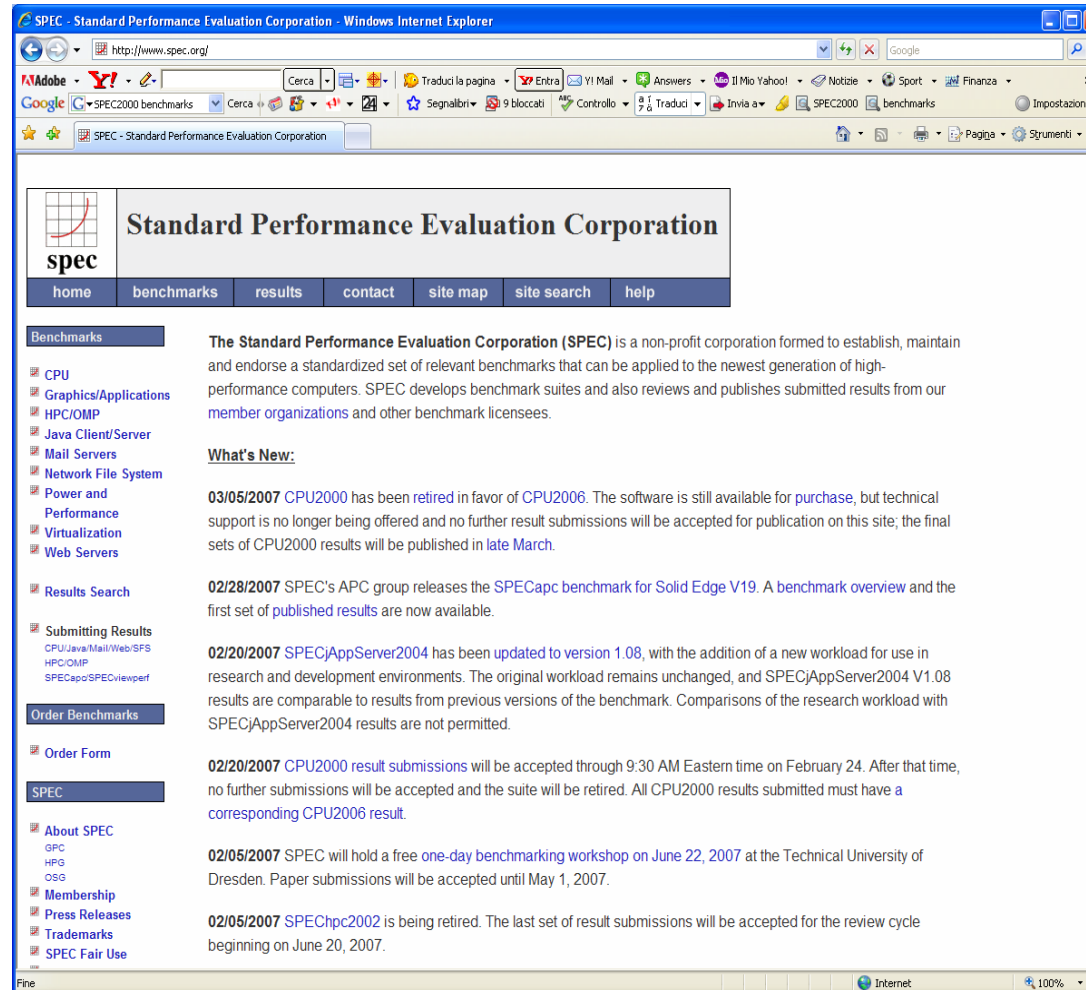
Instruction Set: CISC and RISC

- Complex Instruction Set Computer (CISC)
 - Complex Instructions, es. con operandi in memoria, richiedono un numero di cicli di clock maggiore per eseguire una istruzione (N minore, S maggiore)
- Reduced Instruction Set Computer (RISC)
 - Simple Instructions, minor numero di cicli di clock necessari per eseguire una istruzione (N maggiore, S minore)
 - Semplifica il pipelining

Compilatore

- Traduce un programma espresso in un linguaggio di alto livello in una sequenza di istruzioni macchina
- Un compilatore avanzato sfrutta diverse caratteristiche dell'architettura target al fine di ridurre il numero totale di cicli di clock necessari ad eseguire un programma ($N \cdot S$)
 - Fortemente dipendente dall'architettura del processore
- E' possibile che l'ottimizzazione del codice passi per un riordino delle istruzioni
 - Out of order execution
- Nei linguaggi interpretati, spesso il codice di alto livello è tradotto in un codice simil-assembly che poi viene interpretato da un processore «virtuale» emulato
 - JVM, Java e Bytecode rappresentano un esempio

Performance Measurement



The screenshot shows the SPEC website in a Windows Internet Explorer browser window. The address bar displays <http://www.spec.org/>. The page features the SPEC logo and a navigation menu with links for home, benchmarks, results, contact, site map, site search, and help. The main content area is titled "Benchmarks" and includes a list of benchmark categories: CPU, Graphics/Applications, HPC/OMP, Java Client/Server, Mail Servers, Network File System, Power and Performance, Virtualization, and Web Servers. A "Results Search" section is also present. The "Submitting Results" section lists categories: CPU/Java/Mail/Web/SFS, HPC/OMP, and SPECapp/SPECviewperf. The "Order Benchmarks" section includes an "Order Form" link. The "SPEC" section contains links for "About SPEC" (GFC, HPG, OSG), "Membership", "Press Releases", "Trademarks", and "SPEC Fair Use". The main text area provides information about the Standard Performance Evaluation Corporation (SPEC) and lists several news items with dates and descriptions of benchmark updates and retirements.

Standard Performance Evaluation Corporation

home benchmarks results contact site map site search help

Benchmarks

- CPU
- Graphics/Applications
- HPC/OMP
- Java Client/Server
- Mail Servers
- Network File System
- Power and Performance
- Virtualization
- Web Servers

Results Search

Submitting Results

- CPU/Java/Mail/Web/SFS
- HPC/OMP
- SPECapp/SPECviewperf

Order Benchmarks

- Order Form

SPEC

- About SPEC
 - GFC
 - HPG
 - OSG
- Membership
- Press Releases
- Trademarks
- SPEC Fair Use

The Standard Performance Evaluation Corporation (SPEC) is a non-profit corporation formed to establish, maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers. SPEC develops benchmark suites and also reviews and publishes submitted results from our member organizations and other benchmark licensees.

What's New:

03/05/2007 CPU2000 has been retired in favor of CPU2006. The software is still available for purchase, but technical support is no longer being offered and no further result submissions will be accepted for publication on this site; the final sets of CPU2000 results will be published in late March.

02/28/2007 SPEC's APC group releases the SPECapp benchmark for Solid Edge V19. A benchmark overview and the first set of published results are now available.

02/20/2007 SPECjAppServer2004 has been updated to version 1.08, with the addition of a new workload for use in research and development environments. The original workload remains unchanged, and SPECjAppServer2004 V1.08 results are comparable to results from previous versions of the benchmark. Comparisons of the research workload with SPECjAppServer2004 results are not permitted.

02/20/2007 CPU2000 result submissions will be accepted through 9:30 AM Eastern time on February 24. After that time, no further submissions will be accepted and the suite will be retired. All CPU2000 results submitted must have a corresponding CPU2006 result.

02/05/2007 SPEC will hold a free one-day benchmarking workshop on June 22, 2007 at the Technical University of Dresden. Paper submissions will be accepted until May 1, 2007.

02/05/2007 SPECchpc2002 is being retired. The last set of result submissions will be accepted for the review cycle beginning on June 20, 2007.



Domande di autovalutazione

- Il processore ARM è un processore RISC o CISC? Motivare la risposta
- E il processore Coldfire?
- In un sistema a pipeline una istruzione viene eseguita più velocemente? Motivare la risposta
- Nel Coldfire la long word (4 bytes) **0x871a55af** in memoria è memorizzato:
 - 87 1a 55 af
 - af 55 1a 87