Programmazione I

Lezioni 03-14

Programmazione ed architetture

- La **macchina** comprende un solo tipo di linguaggio, di tipo binario: l'attivazione o meno dei suoi circuiti

  (linguaggio macchina)

- Per renderlo leggibile, al linguaggio macchina è associato un linguaggio di **basso livello** con corrispondenza (sostanzialmente) 1 a 1

  (assembly)

- Le istruzioni sono conservate in memoria e gestite con
  - Fetch
  - Decode
  - Execute

Esempio “ibrido”: Register Machines

- Esperimento “bottom up”: il modello di calcolo astratto delle Register Machines
- Struttura delle Register Machines
- Primitive:
  - INC (r)
  - DEC (r)
  - JNZ (r, etichetta)

  Esempio: creare JZ, J, Reset e Somma usando le operazioni primitive (alla lavagna)
- Discussione: macro e riutilizzo di codice assembly
- Discussione: istruzioni, circuiti, macro, RISC e CISC

Per chi è curioso sulla Turing-equivalenza

- Equivalenza per “emulazione” (es. da assembly a C)

  http://spimsimulator.sourceforge.net/

- … non per forza “decompilazione” (anche se qualche tentativo si può anche fare)!

  https://www.hex-rays.com/products/decompiler/

  http://derevenets.com/

- … e se i linguaggi sono simili anche i tentativi di traduzione pura possono essere sorprendenti!

  ChatGPT, Google BARD … , https://www.mtsystems.com/

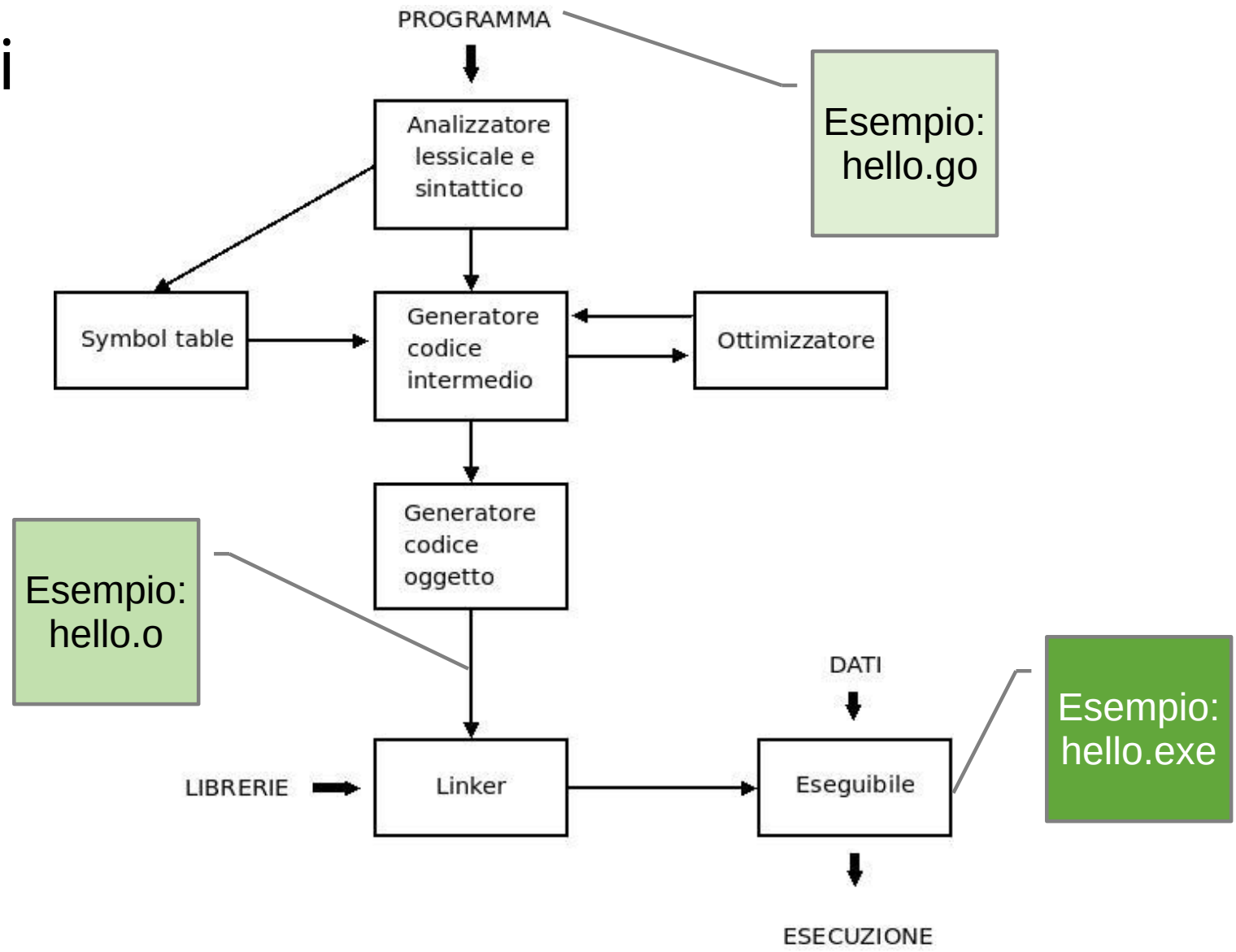
Recap: programmazione ed architetture

- Organizzare la complessità di un problema reale utilizzando istruzioni assembly è **teoricamente possibile**, ma **praticamente impercorribile**.

- Procedimento “top down”

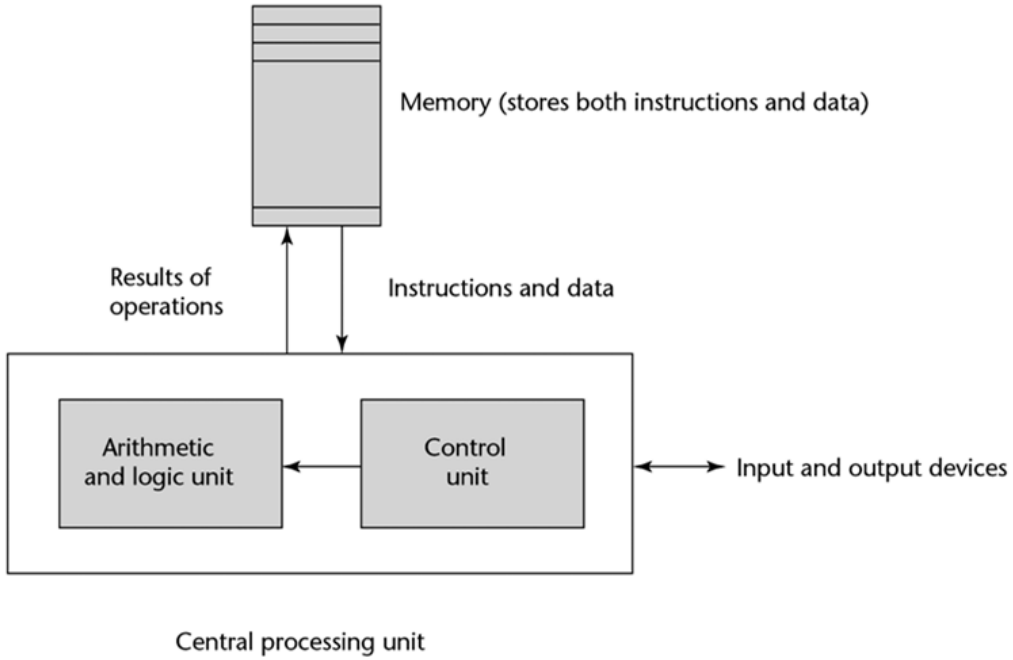
- Ci affidiamo a linguaggi ad **alto livello**, che poi sono tradotti in modo automatico in linguaggi a più basso livello

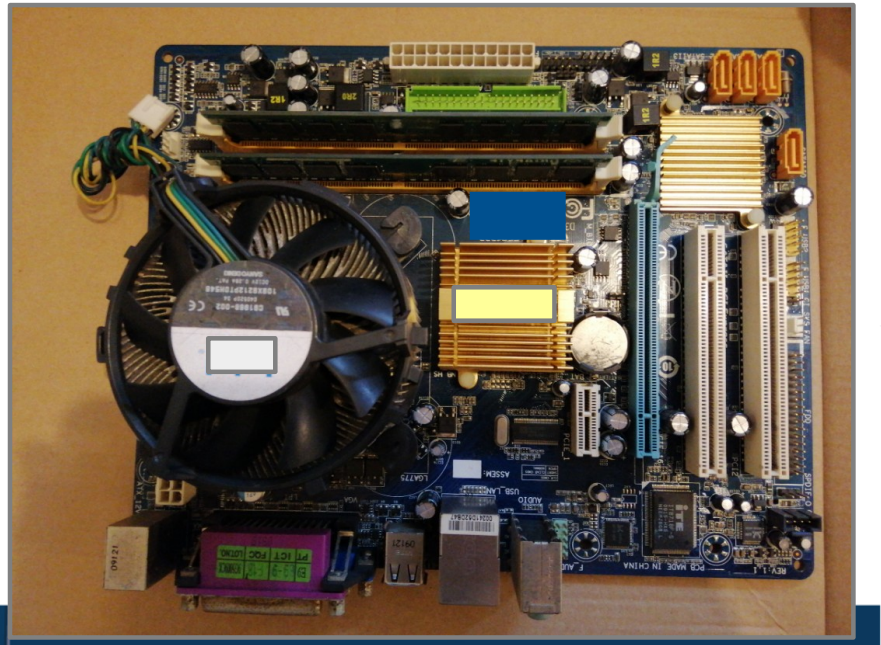
Compilatori



Macchina di Von Neumann

- **Memoria: stato** della macchina
  (i.e. uno snapshot: se ripristinassi tutto ritornerei esattamente alla stessa condizione)
- **CPU:** comandata con **istruzioni**, da eseguire in sequenza





Esecuzione di un programma

- Osservazione: dove sono memorizzati file sorgente, file oggetto, eseguibile (ma anche dati)?
- Fatto: secondo l'architettura di Von Neumann, codice e dati devono essere conservati in RAM
  - Istruzioni:
    - eseguite con ciclo FETCH → DECODE → EXECUTE
  - Dati:
    - portati da RAM ai registri (e viceversa)

Esecuzione di un programma

- Osservazione: dove sono memorizzati file sorgente, file oggetto, eseguibile (ma anche dati)?
- Soluzione: un eseguibile è
  - conservato su disco
  - caricato in RAM al momento dell'avvio (**loading time**)
  - avviato dalla RAM (**run time**)

Dal sorgente all’eseguibile

Per ottenere un’applicazione:

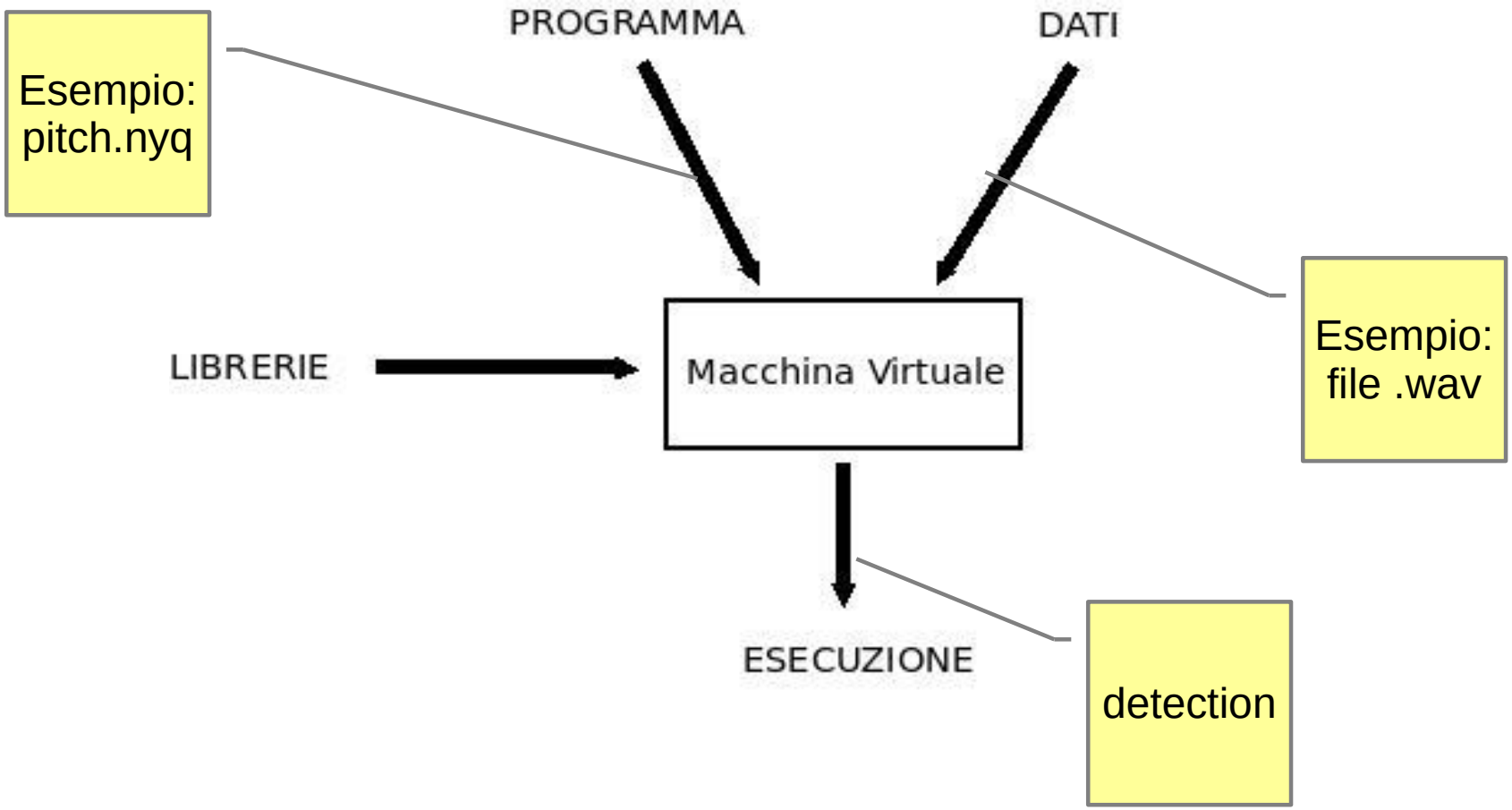
- Editing codice sorgente (programmazione vera e propria)
- Creazione di codice oggetto **(compile time)**
- Creazione eseguibile **(linking time)**
- Esecuzione **(run time)**

Esempio: hello.go

Esempio: visualizzazione assembly hello.o

Esempio: esecuzione hello.exe

Interpreti


Esempio:
pitch.nyq
PROGRAMMA
DATI
LIBRERIE
Macchina Virtuale
Esempio:
file .wav
ESECUZIONE
detection

Data abstraction

```
1000: lw   $2, <2000>
1004: add  $3, $0, $0
1008: add  $4, $3, $0
1012: mult $4, $4, $4
1016: slt  $5, $4, $2
1020: beq  $5, $0, <1032>
1024: addi $3, $3, +1
1028: jump <1008>
1032: sw   $3, <3000>
```

L’idea di variabile

- Astrazione di una (o più) celle di memoria
- Esempio: uso di variabili in un ambiente interattivo (python)
  - Il concetto di assegnamento: valori a variabili
  - Lettura di valori in variabili
  - Ancora assegnamento: variabili a variabili
  - Espressioni con variabili
  - Variabili e numeri (interi, floating)

Il primo programma GO

- Storia di GO e ragioni di adozione
  - Solide basi di design (Google - 2007/2009)
    - R. Griesemer → V8 JavaScript engine, HotSpot Java VM
    - R. Pike → UNIX, Plan9, Inferno
    - K. Thompson → UNIX, B, ed, UTF-8, regex
  - Fully open source
  - Costantemente buon “ranking”
  - Efficienza
  - Gestione “safe” di tipi e memoria
  - golang.org
    - Tutorial, download, playground, documentazione
    - https://golang.org/pkg/ (GO standard library)

Il primo programma GO

- Esempio: costruzione di un sorgente (pattern #0), “pattern_0.go”
- Variabili in GO
  - Dichiarazione di una variabile
    - Keyword
    - Identificatore
    - Tipo
  - Assegnamento
  - Copia tra variabili
  - Operazioni algebriche

Dal sorgente all'*esecuzione*

Da terminale ...

- Editing del codice sorgente e run simultaneo

Esempio: hello.py

Recap: astrazione!

- Assembly →

  Linguaggio di Programmazione ad alto livello (GO)

- Process abstraction:

  ADD(Rd, Ra, Rb) *(def)*← INC, DEC, JZ

- Data abstraction:

  Il concetto di variabile

  Esperimento: variabili in un interprete

La vita di una variabile

1 - Dichiarazione

2 - Definizione (inizializzazione o allocazione)

3 - Utilizzo

4 - Rilascio

Il primo programma GO

- Rispetto di **regole** a diversi **livelli** (idea generale)
  - Lessicale
  - Sintattico
  - Semantico
  - *Estetico (o pragmatico)*

Livello lessicale

- Regole di combinazione di singoli simboli
  → per produrre “elementi base” (token) validi
- Esempi:

```
index,  =,  +,  2,  3
```

  - Separatori e whitespace
  - Tipi di token:
    - Keyword
    - Identificatori (predefiniti / dichiarati dal programmatore)
    - Separatori (parentesi, virgole …)
    - Operatori
    - Letterali (costanti)

Livello sintattico

- Regole di combinazione degli elementi lessicali base (validi)
  → per produrre istruzioni valide
- Esempi:

```
index =  2  +  3
<id>  =  <espr.>
```

  - Dichiarazione di “package”
  - Prototipo di una funzione

Livello semantico

- Significato (o “effetto”) delle istruzioni
  (e del programma in generale)
- Esempi:

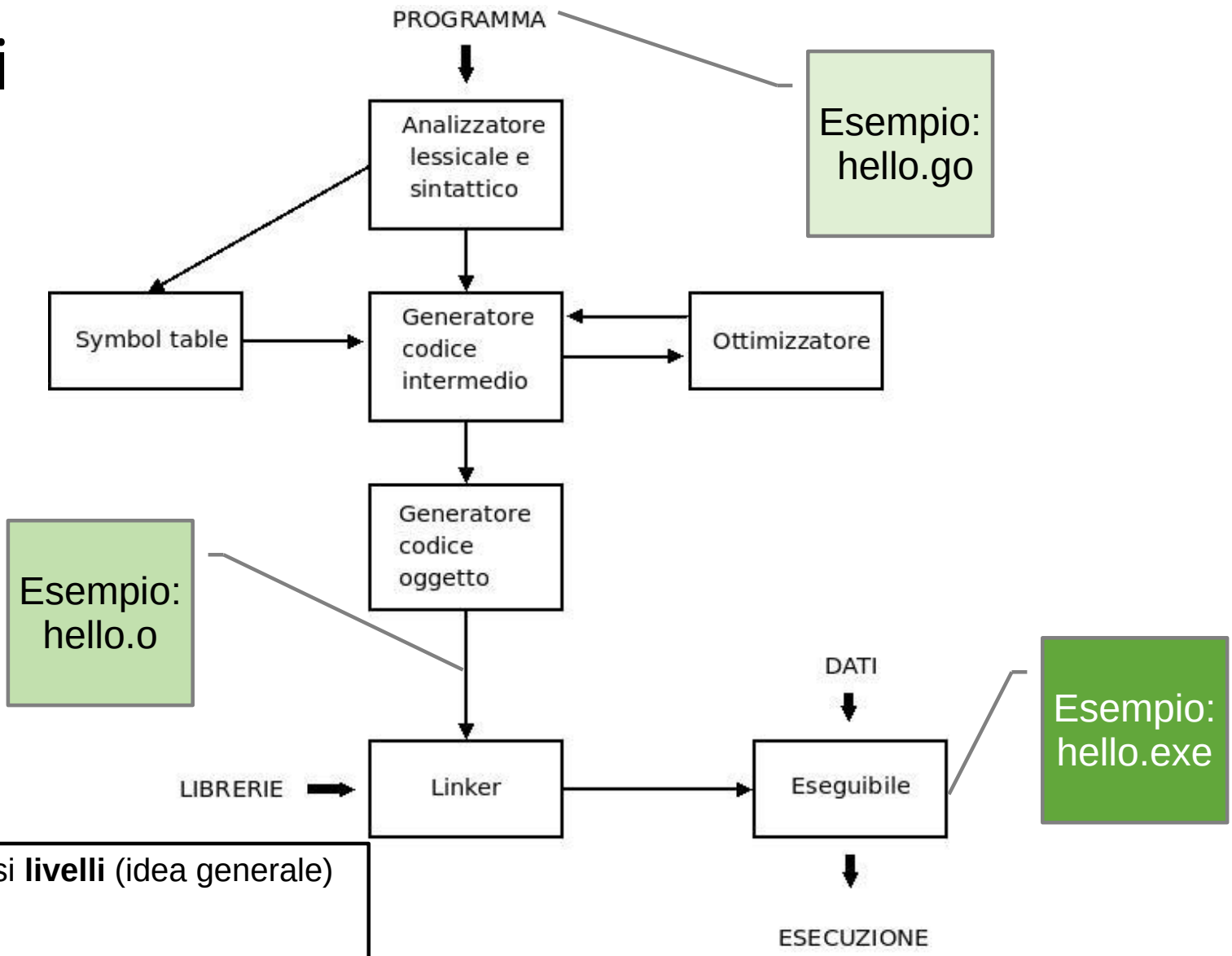
```
(index = 2 + 3)
esegui la somma delle costanti 2 e 3, e conserva il
risultato con l’identificatore “index”
```

  - Tema ricorrente nelle prossime lezioni …

Livello pragmatico/estetico

- Esempi:
  - Scelta di dove andare a capo
  - Distribuzione del testo (indentazione)
  - *Scelta dei nomi degli identificatori*

Compilatori



- Rispetto di **regole** a diversi **livelli** (idea generale)
  - Lessicale
  - Sintattico
  - Semantico
  - *Estetico (o pragmatico)*

Formalmente le variabili:

- Caratteristiche
  - Identificatore (nome)
  - Valore
  - Tipo
  - Scope
  - *Indirizzo*
- Definizione di tipo
  1. l'insieme di valori che una variabile può assumere
  2. le operazioni ammesse su quella variabile
- Definizione di scope (tra qualche lezione)

Formalmente le variabili:

- Caratteristiche
  - Identificatore (nome)
  - Valore
  - Tipo
  - Scope
  - *Indirizzo*
- Definizione di tipo
  1. l'insieme di valori che una variabile può assumere
  2. le operazioni ammesse su quella variabile
- Definizione di scope (tra qualche lezione)

Il sistema dei tipi

- Tipi “base” in GO
  - `bool`
  - `string`
  - `int  int8  int16  int32  int64`
  - `uint uint8 uint16 uint32 uint64 uintptr`
  - `byte // alias for uint8`
  - `rune // alias for int32, represents a Unicode code point`
  - `float32 float64`
  - `complex64 complex128`
- int e uint sono “implementation dependent” (32 o 64 bit)

Il sistema dei tipi

- Tipi “base” in GO
  - `bool`
  - `string`
  - **`int  int8  int16  int32  int64`**
  - **`uint uint8 uint16 uint32 uint64`** `uintptr`
  - `byte // alias for uint8`
  - `rune // alias for int32, represents a Unicode code point`
  - `float32 float64`
  - `complex64 complex128`
- int e uint sono “implementation dependent” (32 o 64 bit)

Operatori in GO

Binari						
*	/	%	<<	>>	&	&^
+	-	|	^			
==	!=	<	<=	>	>=	
&&						
||						
Unari						
++	--					
+	-	!	^	*	&	<-

Operatori in GO

Binari						
*	/	%	<<	>>	&	&^
+	-	|	^			
==	!=	<	<=	>	>=	
&&						
||						
Unari						
++	--					
+	-	!	^	*	&	<-

Recap: il sistema dei tipi

- Tipi “base” in GO
  - `bool`
  - `string`
  - `int  int8  int16  int32  int64`
  - `uint uint8 uint16 uint32 uint64 uintptr`
  - `byte // alias for uint8`
  - `rune // alias for int32, represents a Unicode code point`
  - **`float32 float64`**
  - `complex64 complex128`

Operatori in GO

Binari						
*	/	%	<<	>>	&	&^
+	-	|	^			
==	!=	<	<=	>	>=	
&&						
||						
Unari						
++	--					
+	-	!	^	*	&	<-

Strong typing in GO

- Esperimenti sulla filosofia “strong typing” di GO
- Il meccanismo di conversione temporanea di tipo (***cast e coercion)***

Recap: il sistema dei tipi

- Tipi “base” in GO
  - `bool`
  - `string`
  - `int  int8  int16  int32  int64`
  - `uint uint8 uint16 uint32 uint64 uintptr`
  - `byte // alias for uint8`
  - `rune // alias for int32, represents a Unicode code point`
  - `float32 float64`
  - `complex64 complex128`

Operatori in GO

Binari						
*	/	%	<<	>>	&	&^
+	-	|	^			
==	!=	<	<=	>	>=	
&&						
||						
Unari						
++	--					
+	-	!	^	*	&	<-

Operatori in GO

Binari						
*	/	%	<<	>>	&	&^
+	-	|	^			
==	!=	<	<=	>	>=	
&&						
||						
Unari						
++	--					
+	-	!	^	*	&	<-

Tipo bool

- Ripasso: algebra booleana (not, and, or, leggi di de Morgan)
- Tipi bool in GO (codice)
- Esperimenti sulla filosofia “strong typing” di GO

- Operazioni su variabili bool, con risultato bool
- Operazioni su variabili numeriche, con risultato bool (confronti)

Le costanti

- Idea:
  - valori (literals) con particolare significato
  - Ricorrono piu` volte nel programma
  - Esempio: valore di π
  - Alcune già *predefinite* in GO (esempio math.Pi, math.MaxInt32)
- Esempio: dichiarazione di costanti in GO

Process abstraction

```
1000: lw   $2, <2000>
1004: add  $3, $0, $0
1008: add  $4, $3, $0
1012: mult $4, $4, $4
1016: slt  $5, $4, $2
1020: beq  $5, $0, <1032>
1024: addi $3, $3, +1
1028: jump <1008>
1032: sw   $3, <3000>
```

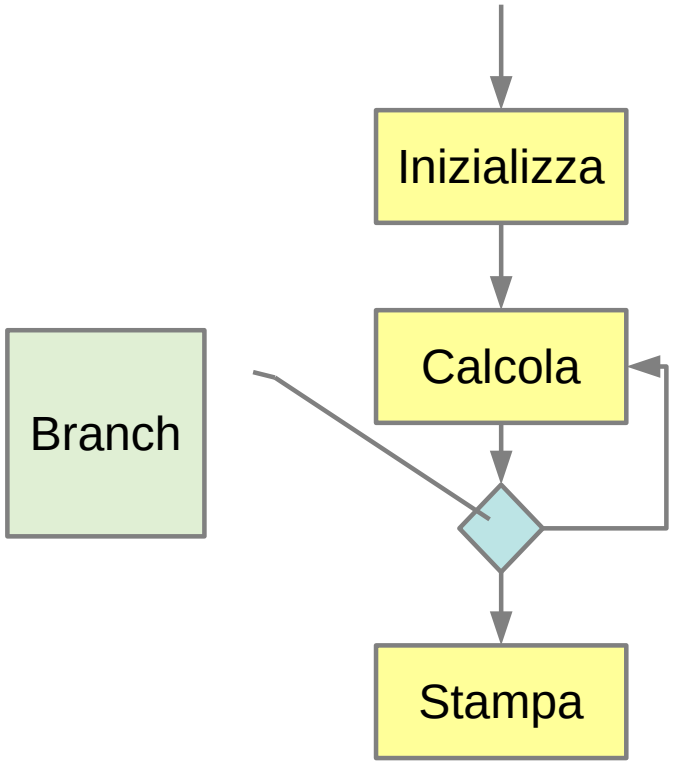
Il controllo del flusso

- Recap: PC, IR e istruzioni JUMP
  (discussione su codice assembly per il calcolo della radice quadrata)
- Idea: ottenere lo stesso effetto, senza JUMP

Un modello per il controllo di flusso

Un modello per il controllo del flusso

- Il formalismo dei diagrammi a blocchi



Strumenti per flowchart

- Flowgorithm: http://www.flowgorithm.org/
- Algobuild: https://www.algobuild.com
- Raptor: https://raptor.martincarlisle.com/

Il controllo del flusso

- Costrutto #1: la **sequenza** (esempio radq, esempio in GO)
- Costrutto #2: la **selezione binaria**

if <condizione (vero/falso)>
then <sequenza di istruzioni (se cond. vera) >
else <sequenza di istruzioni (se cond. falsa) >

- if – then – else in GO (if_then_else.go)
- Note sulla sintassi: obbligatorietà e posizione delle graffe

Esercizi

- Stabilisci se un numero è pari o dispari
- Stampa il minore
- Individua numeri negativi
- Dato il voto di uno studente, stampa l’esito dell’esame (ripeti, promosso, orale facoltativo)

L’attività di “programmazione”

- Calcolo delle radici di un’equazione di secondo grado
  - Analisi
  - Intuizione e casi limite, casi di test
  - Stesura delle Specifiche
  - Progettazione del metodo risolutivo
  - Formalizzazione attraverso flow chart
  - Scrittura di codice GO
  - Testing e Debugging

Ciclo di vita del software

- Analisi delle esigenze
- Studio di fattibilità

  1) Stesura delle specifiche

  2) Progettazione

  3) Sviluppo

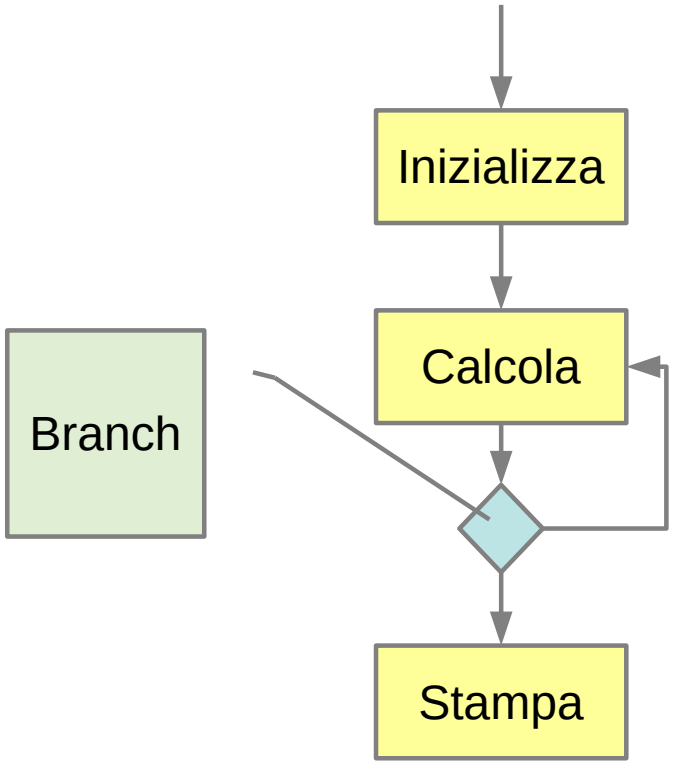
  4) Testing e Debugging (goto 3 o anche 2)

  5) Rilascio (goto 1)

  6) Uso, aggiornamento e manutenzione

Un modello per il controllo del flusso

- Esercizio: calcolo della radice quadrata in GO



Programmazione I

Lezione 9

Recap: controllo del flusso

- Formalismo dei diagrammi a blocchi
- Esempio: esperimento “inverso” con codice per rad.q.
- Costrutto #1: la **sequenza** (esempio radq, esempio in GO)
- Costrutto #2: la **selezione binaria**

if <condizione (vero/falso)>
then <sequenza di istruzioni (se cond. vera) >
else <sequenza di istruzioni (se cond. falsa) >

Dalla selezione ai cicli

- Idea (sviluppata su esempio di controllo dell'input):
  - Parto da un costrutto di selezione if-then-else …
  - … in particolare, la versione con ramo else vuoto

    if <condizione> **{**

    <sequenza>

    }
  - Rendo il costrutto "iterativo": al termine della sequenza lo ripeto, partendo dalla valutazione della condizione

    for <condizione> **{**

    <sequenza>

    }
  - Osservo che ho sempre tre ingredienti:
    (1) inizializzazione (2) condizione (3) aggiornamento

Esercizi

- Stampare la somma delle cifre di un numero di 3 cifre
- Stabilire se la somma delle prime tre cifre di un numero sia < 10
- Leggi numeri (di quantità nota) e stampa la somma
- Leggi numeri e stampa la media
- Leggi numeri e somma fino a trovare uno 0

Il controllo del flusso: costrutto di iterazione

- Costrutto #3: il **ciclo**
- Forma “unaria”

  for <condizione> **{**

  <sequenza>

  }
- Forma “ternaria”

  for <init>; <condizione>; <aggiornamento> **{**

  <sequenza>

  }
- Forma “zero-aria”: **for {** <sequenza> **}**

Esempi

[U] Stampa “Ciao!” un certo numero di volte

[T] Conto alla rovescia

Esercizi

- Stampare la somma delle cifre di un numero di 3 cifre
- Stabilire se la somma delle prime tre cifre di un numero sia < 10
- Leggi numeri (di quantità nota) e stampa la somma
- Leggi numeri e stampa la media
- Leggi numeri e somma fino a trovare uno 0

Esempi

- Media di N numeri
- Uscita da un ciclo con valore “tappo”
  - Esempio: leggi input da utente, e fermati quando viene inserito il valore 0 (for senza argomenti)
  - Esempio: calcola la media di una serie di numeri in ingresso
- Break e continue:
  - Esempio: media di una sequenza di numeri terminata da 0
  - Esempio: media di una sequenza di numeri terminata da 0, saltando i negativi
- Sequenze e strutture di controllo annidate
  - Cicli for innestati: esempio della scacchiera