

Corso di Architettura degli Elaboratori e Laboratorio (M-Z)

Esercitazione ARM 1

Nino Cauli



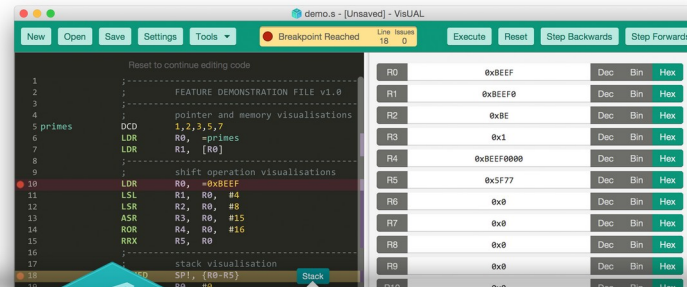
UNIVERSITÀ
degli STUDI
di CATANIA

Dipartimento di Matematica e Informatica

- Durante le esercitazioni in assembly (ARM) useremo l'emulatore VisUAL:

<https://salmanarif.bitbucket.io/visual/index.html>

- Insieme di istruzioni ridotto:
https://salmanarif.bitbucket.io/visual/supported_instructions.html
- Visualizzazione grafica contenuto registri, bit di stato e memoria
- Debug mode sia in modalità breakpoint che in modalità trace



VisUAL
A highly visual ARM emulator

name **DCD** lista-di-parole

- Usato per dichiarare una o più parole in memoria (DATAWORD)

name **FILL** bytes-da-riservare

- Riserva uno spazio in memoria vuoto (RESERVE)

END

- Termina l'emulazione (END)

- Registro: **Ri**
- Diretto: **Locazione**
- Indiretto: **[Ri]**
- Immediato: **#Valore**
- Base e spiazzamento: **[Ri, #Valore]**
- Pre-base e spiazzamento: **[Ri, #Valore]!** Generalizza auto-decremento
- Post-base e spiazzamento: **[Ri], #Valore** Generalizza auto-incremento

LDR Ri, [Rj]

- Carica nel registro Ri il contenuto della parola di memoria puntata da Rj

STR Ri, [Rj]

- Salva nella parola di memoria puntata da Rj il contenuto di Ri

MOV Ri, Rj / MOV Ri, #Valore

- Carica nel registro Ri il contenuto di Rj o il valore immediato

MVN Ri, Rj / MVN Ri, #Valore

- Uguale a MOV solo che prima di caricare il valore in Ri lo nega

ADD Rd, Ri, Rj

- Somma il contenuto di Ri e Rj e lo carica nel registro Rd (non aggiorna i bit di stato (NZCV))

SUB Rd, Ri, Rj

- Sottrae il contenuto di Rj da Ri e lo carica nel registro Rd (non aggiorna i bit di stato (NZCV))

ADDS Rd, Ri, Rj

- Somma il contenuto di Ri e Rj e lo carica nel registro Rd (**aggiorna** i bit di stato (NZCV))

SUBS Rd, Ri, Rj

- Sottrae il contenuto di Rj da Ri e lo carica nel registro Rd (**aggiorna** i bit di stato (NZCV))

CMP R_i, R_j

- Sottrae R_j da R_i e aggiorna i bit di stato. Il risultato viene scartato

CMN R_i, R_j

- Somma R_i e R_j e aggiorna i bit di stato. Il risultato viene scartato

B Indirizzo-salto

- Salto incondizionato

Bcodice Indirizzo-salto

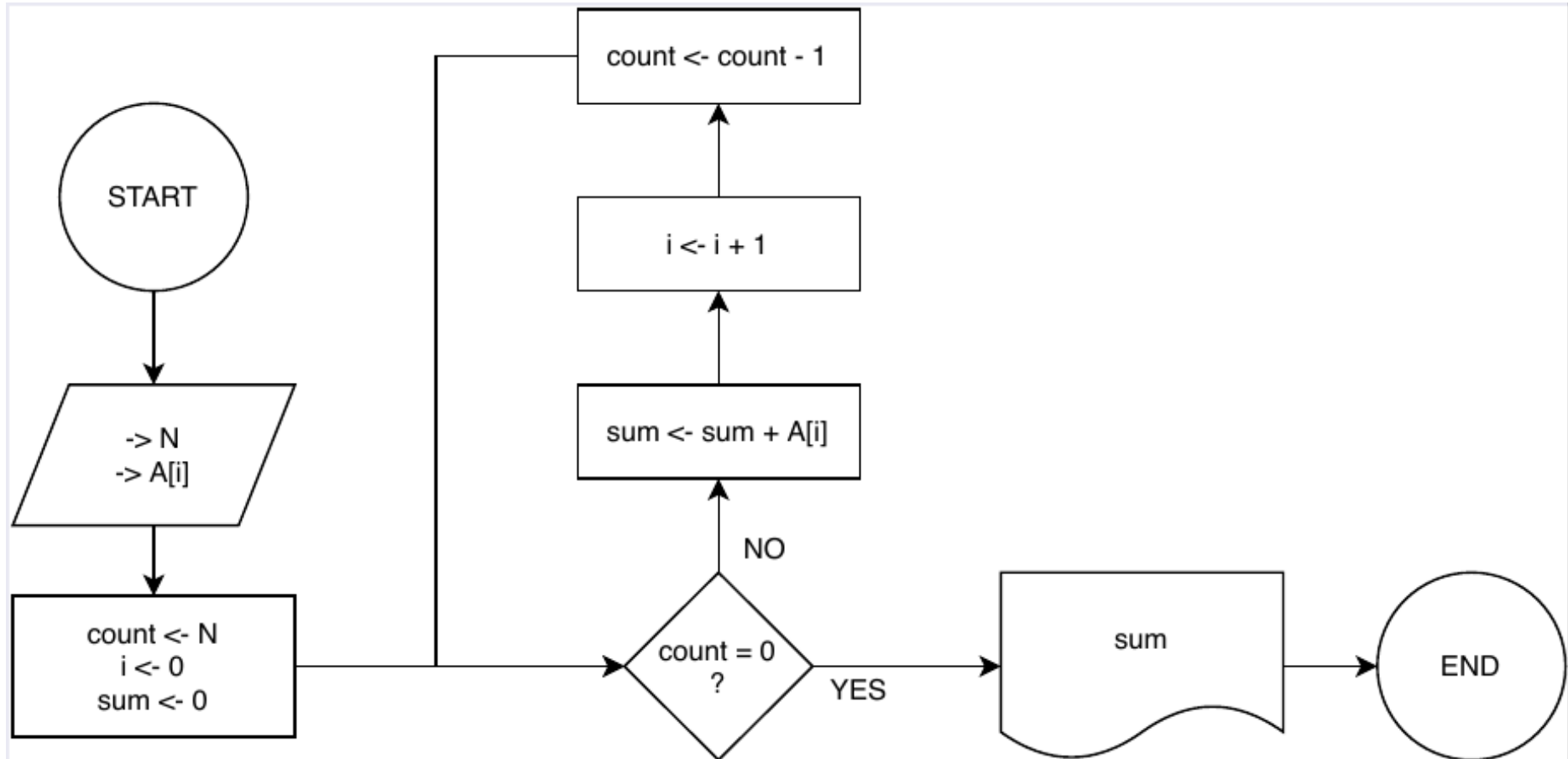
- Salto condizionato che valuta la condizione definita dal suffisso sulla base dei bit di stato

Codici di condizione sui bit di stato

Bit di stato	Significato
N (negativo)	1 se risultato negativo, 0 se positivo o nullo
Z (zero)	1 se risultato nullo, 0 altrimenti
V (trabocco)	1 se trabocco in comp. a due, 0 altrimenti (oVerflow)
C (riporto)	1 se trabocco in binario naturale, 0 altrimenti (Carry)

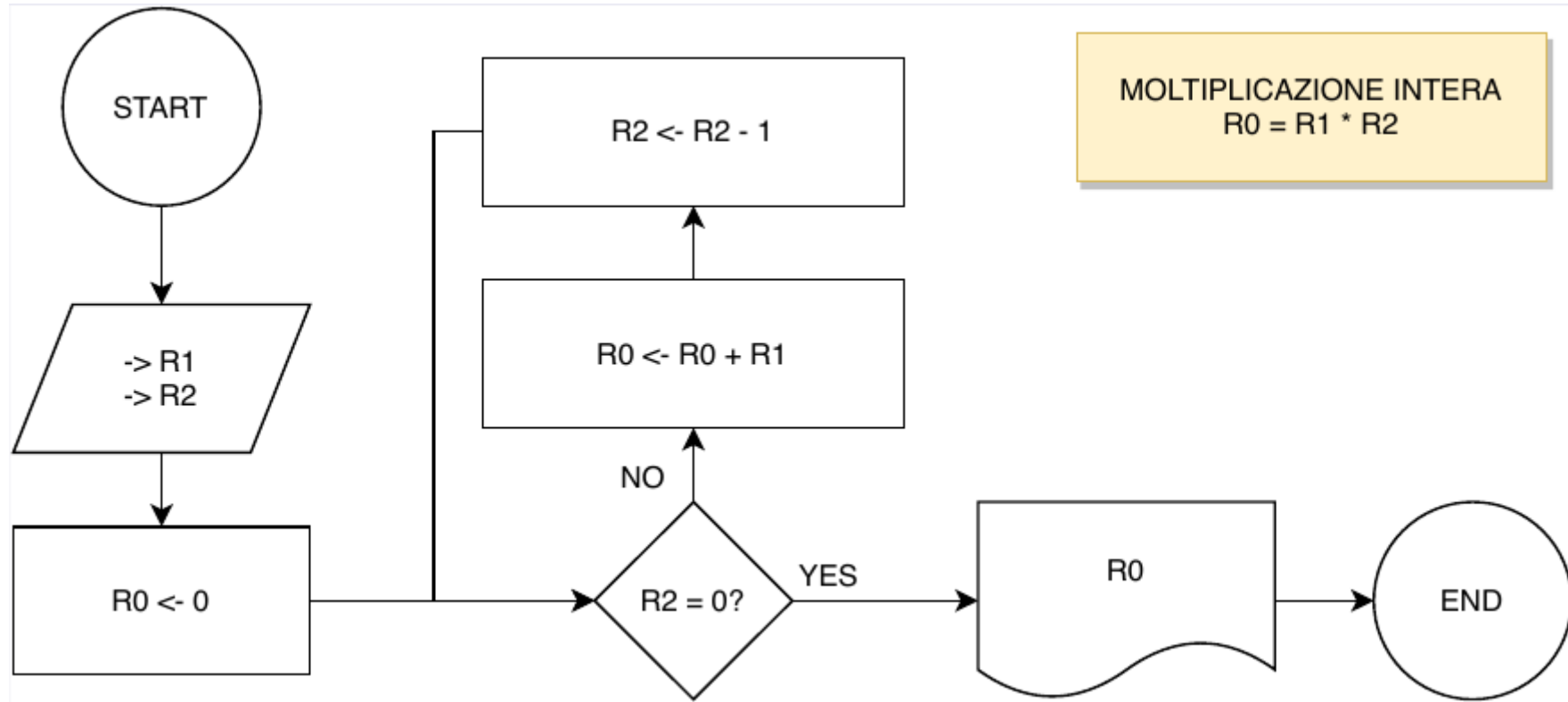
Codice	Significato
EQ (equal)	Z
NE (not equal)	$\neg Z$
GE (greater-equal)	$\neg(N \oplus V)$
LT (lower than)	$N \oplus V$
GT (greater than)	$\neg(Z \text{ OR } (N \oplus V))$
LE (lower-equal)	$Z \text{ OR } (N \oplus V)$

Somma di n numeri

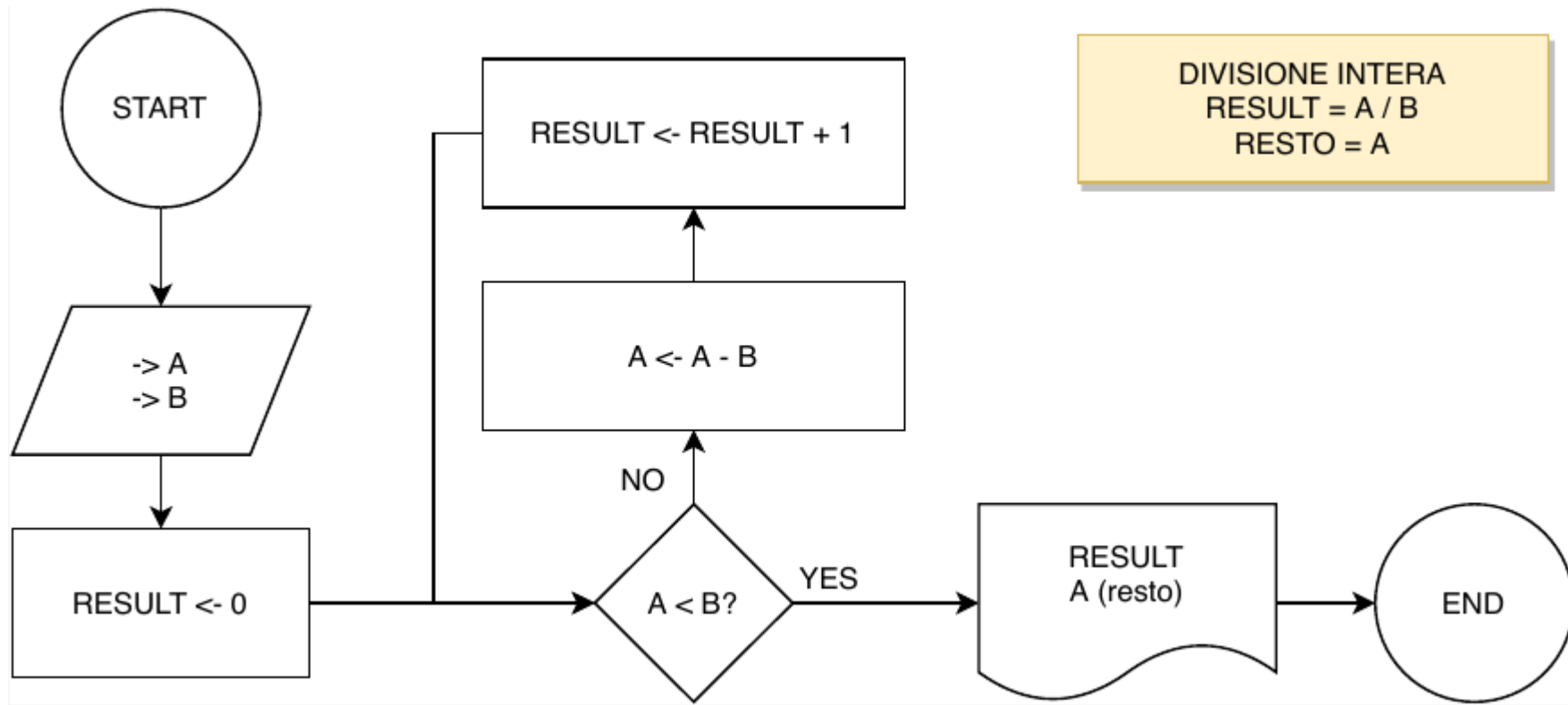


NUMEL	DCD	5
NUM1	DCD	2,3,4,1,5
SOMMA	FILL	4
	MOV	R6, #SOMMA
	MOV	R2, #NUMEL
	LDR	R2, [R2]
	MOV	R3, #NUM1
CICLO	LDR	R5, [R3]
	ADD	R4, R4, R5
	ADD	R3, R3, #4
	SUBS	R2, R2, #1
	BGT	CICLO
	STR	R4, [R6]
	END	

Moltiplicazione intera



FAT1	DCD	5
FAT2	DCD	7
RIS	FILL	4
	MOV	R2, #FAT1
	LDR	R2, [R2]
	MOV	R3, #FAT2
	LDR	R3, [R3]
	MOV	R5, #RIS
	MOV	R4, #0
CICLO	CMP	R3, #0
	BEQ	FINE
	SUB	R3, R3, #1
	ADD	R4, R4, R2
	B	CICLO
FINE	STR	R4, [R5]
	END	



DIVID	DCD	274
DIVIS	DCD	13
RIS	FILL	4
REST	FILL	4
	MOV	R2, #DIVID
	LDR	R2, [R2]
	MOV	R3, #DIVIS
	LDR	R3, [R3]
	MOV	R5, #RIS
	MOV	R6, #REST
	MOV	R4, #0
CICLO	CMP	R2, R3
	BLT	FINE
	SUB	R2, R2, R3
	ADD	R4, R4, #1
	B	CICLO
FINE	STR	R4, [R5]
	STR	R2, [R6]
	END	

- Calcolo del massimo e del minimo
- Ricerca di un elemento in un array
- Conteggio di numero di occorrenze di un elemento in un array
- Calcolo del prodotto scalare di due vettori di uguale dimensione
- Ordinamento di un array (Bubblesort)

- 3 registri speciali:
- **R13 = SP**
- **R14 = LR**
- **R15 = PC**
- Chiamata a sottoprogramma:
- **BL** *IndirizzoSottoprogramma* (salva il PC nel LR prima di effettuare il salto)
- Rientro da sottoprogramma:
- **MOV** *PC, LR*

- Per impilare e spilare dati si possono usare le istruzioni di load multiple e store multiple
- POP:
- **STM[codice]** *StackPointer[!], {lista di registri}* (registro di ordine più basso in cima)
- PUSH:
- **LDM[codice]** *StackPointer[!], {lista di registri}* (registro di ordine più basso dalla cima)
- Codice e !:
- **Codice:** *FD* = full descending, *FA* = full ascending, *ED* = empty descending, *EA* = empty ascending
- **!** = aggiorna il valore del puntatore alla cima a fine operazione