ARCHITETTURA DEGLI ELABORATORI

A.A. 2020-2021

Università di Napoli Federico II
Corso di Laurea in Informatica

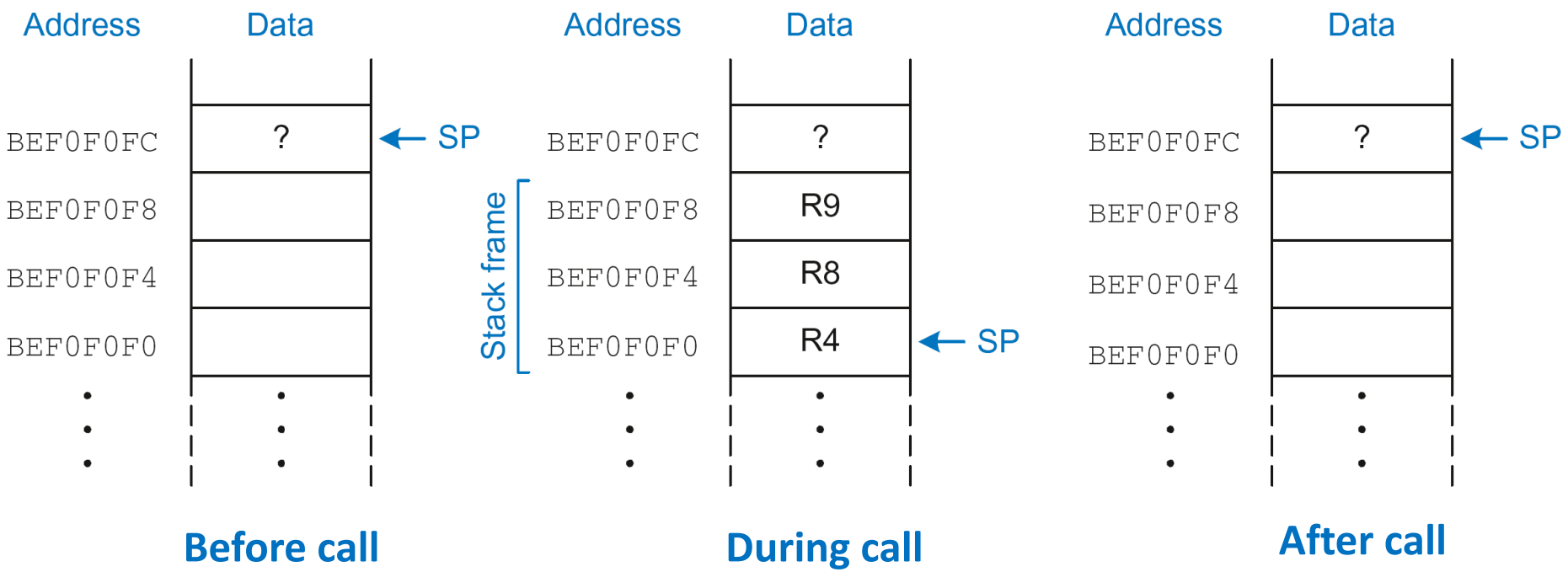
Docenti
Proff. Luigi Sauro gruppo 1 (A-G)
Silvia Rossi gruppo 2 (H-Z)

Storing Register Values on the Stack

ARM Assembly Code

```
; R2 = result
DIFFOFSUMS
  SUB SP, SP, #12     ; make space on stack for 3 registers
  STR R4, [SP, #8]    ; save R4 on stack
  STR R8, [SP, #4]    ; save R8 on stack
  STR R9, [SP]        ; save R9 on stack
  ADD R8, R0, R1      ; R8 = f + g
  ADD R9, R2, R3      ; R9 = h + i
  SUB R4, R8, R9      ; result = (f + g) - (h + i)
  MOV R0, R4          ; put return value in R0
  LDR R9, [SP]        ; restore R9 from stack
  LDR R8, [SP, #-4]   ; restore R8 from stack
  LDR R4, [SP, #-8]   ; restore R4 from stack
  ADD SP, SP, #12     ; deallocate stack space
  MOV PC, LR          ; return to caller
```

The Stack during `diffofsums` Call



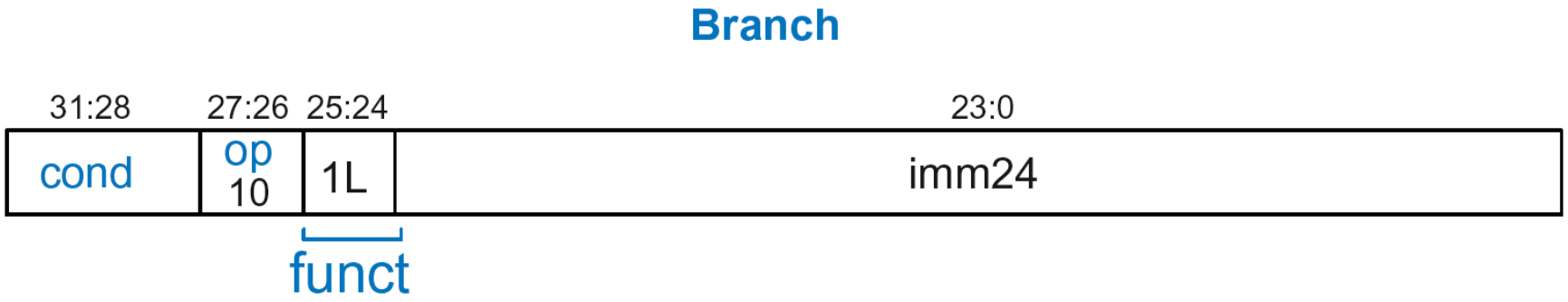
Instruction Formats

- Data-processing
- Memory
- **Branch**

Branch Instruction Format

Encodes `B` and `BL`

- ***op*** = 10_2
- ***imm24*****:** 24-bit immediate
- ***funct*** = $1L_2$: ***L*** = 1 for `BL`, ***L*** = 0 for `B`



Encoding Branch Target Address

- ***Branch Target Address* (BTA)*:*** Next PC when branch taken
- BTA is relative to current PC + 8
- *imm24* encodes BTA
- ***imm24*** = # of words BTA is away from PC+8

Branch Instruction: Example 1

ARM assembly code

```
0xA0          BLT THERE          ← PC
0xA4          ADD R0, R1, R2
0xA8          SUB R0, R0, R9     ← PC+8
0xAC          ADD SP, SP, #8
0xB0          MOV PC, LR
0xB4  THERE   SUB R0, R0, #1     ← BTA
0xB8          BL  TEST
```

- PC = 0xA0
- PC + 8 = 0xA8
- THERE label is 3 instructions past PC+8
- So, *imm24* = 3

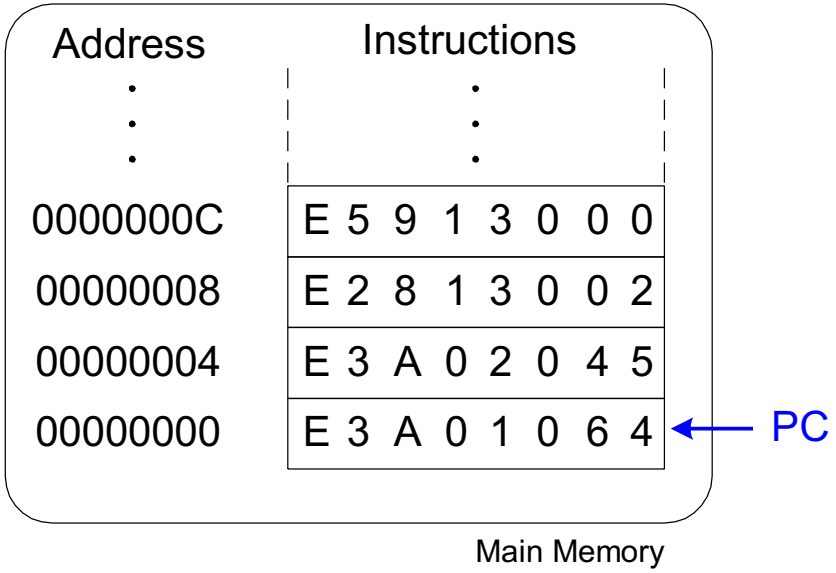
Power of the Stored Program

- **32-bit instructions & data** stored in memory
- **Sequence of instructions:** only difference between two applications
- **To run a new program:**
  - No rewiring required
  - Simply store new program in memory
- **Program Execution:**
  - Processor *fetches* (reads) instructions from memory in sequence
  - Processor performs the specified operation

The Stored Program

Assembly Code	Machine Code
`MOV R1, #100`	`0xE3A01064`
`MOV R2, #69`	`0xE3A02045`
`ADD R3, R1, R2`	`0xE2813002`
`STR R3, [R1]`	`0xE5913000`

Stored Program



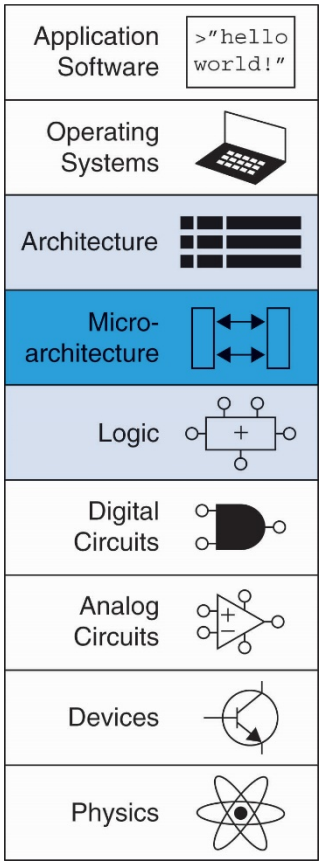
Program Counter (PC): keeps track of current instruction

MICROARCHITETTURA ARM

Introduction

Architettura: rappresenta la struttura di un calcolatore dal punto di vista di un programmatore

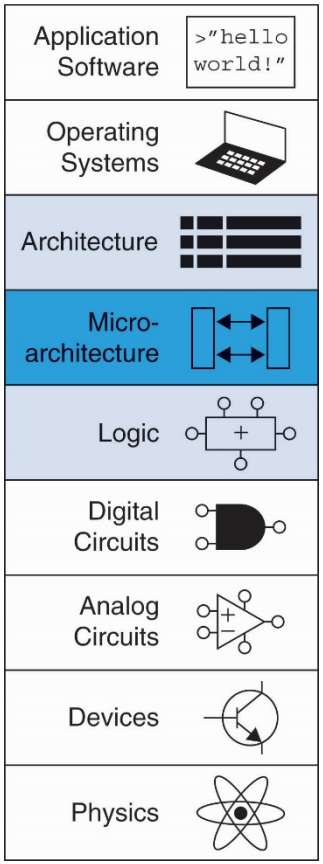
- È definita dal set di istruzioni e operandi che costituisce il *linguaggio macchina*
- Al linguaggio macchina corrisponde un linguaggio *assembly*
- Esistono molte architetture differenti
  - **ARM**
  - **X86**
  - **MIPS**
  - **SPARC**
  - **PowerPC**



Introduction

Microarchitettura: definisce la disposizione specifica di registri, ALU, macchine a stati finiti, le memorie e altri blocchi logici necessari per implementare una architettura

- Come due algoritmi di ordinamento (bubble e quick sort) realizzano la stessa funzione con procedure differenti, così due microarchitetture possono realizzare la medesima architettura ma soluzioni tecnologiche e prestazioni molto differenti
- Ad esempio Intel e AMD realizzano diversi modelli (microarchitetture) della medesima architettura x86.

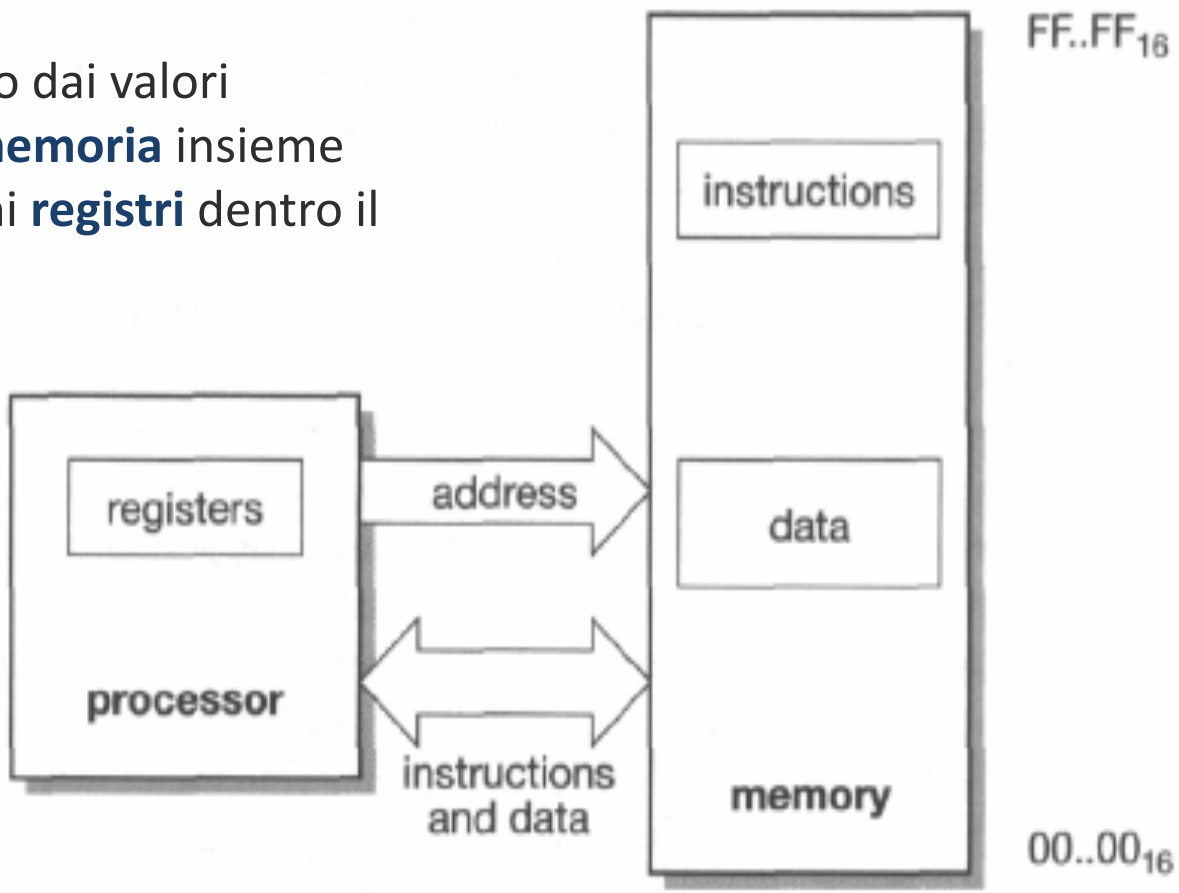


ARM Cos'è un processore

Un processore general-purpose è un **automa** a stati finiti, che esegue **istruzioni** rilocate in una memoria.

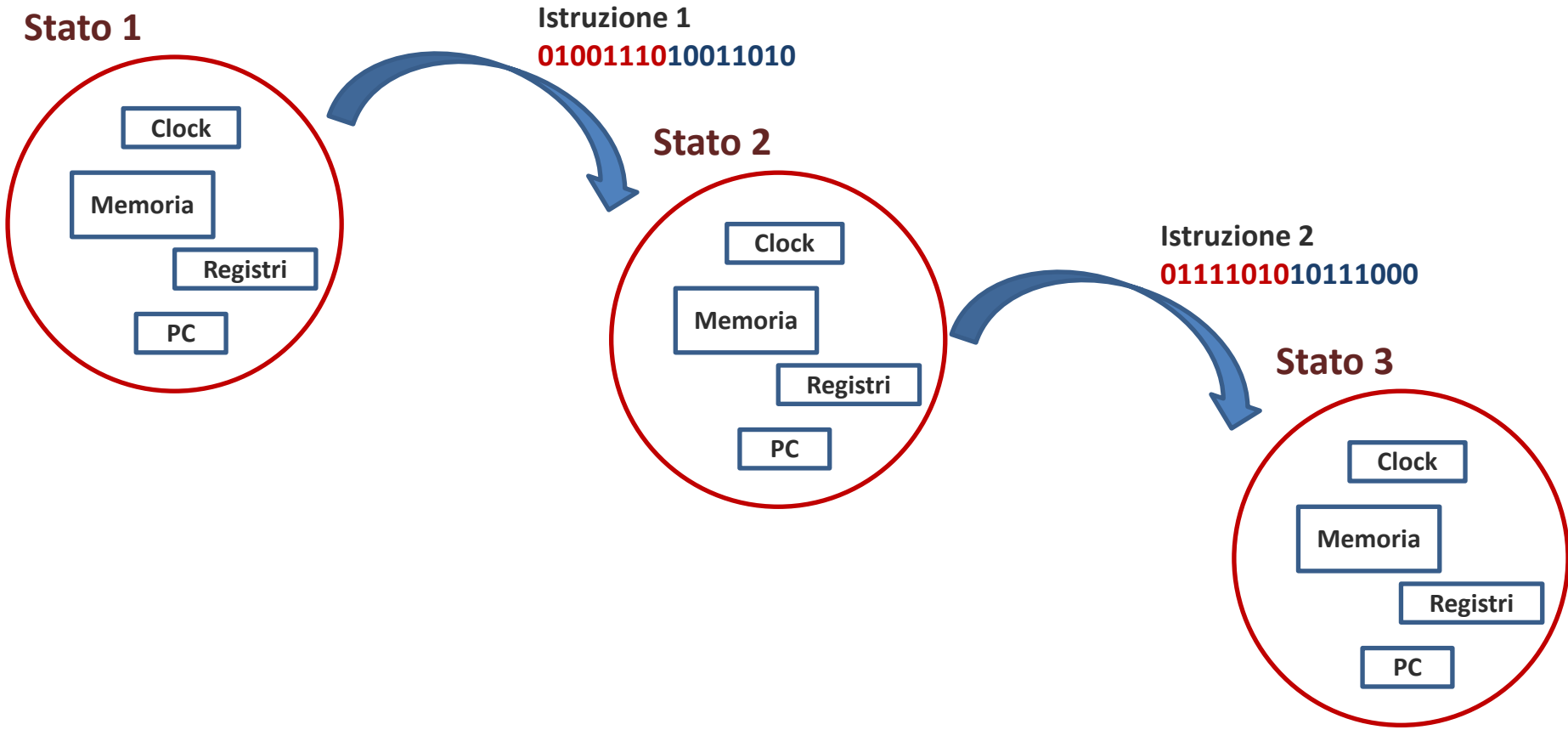
Lo **stato** del sistema è definito dai valori contenuti nelle locazioni di **memoria** insieme con i valori contenuti in alcuni **registri** dentro il processore stesso.

Ogni **istruzione** definisce in che modo lo stato deve cambiare e quale istruzione deve essere eseguita successivamente.



Microarchitetture come automi

Una microarchitettura può essere vista come un automa.



Variazioni di stato

- I registri, la memoria dati e la memoria istruzioni operano mediante logica combinatoria.
- Tutti i componenti effettuano la scrittura sul fronte alto del clock, cosicché lo stato del sistema cambia solo su un fronte del clock.
- Gli indirizzi, i dati ed il segnale di write enable devono essere impostati prima del fronte del clock e mantenuti immutati per un tempo superiore al ritardo di propagazione.
- Sia gli elementi di stato, che il microprocessore sono costituiti da logica combinatoria e da componenti sincronizzate dal clock. L'intero sistema è, quindi, sincrono e può essere visto come una complessa macchina a stati finiti o come l'insieme di macchine a stati finiti semplici, che interagiscono fra loro.

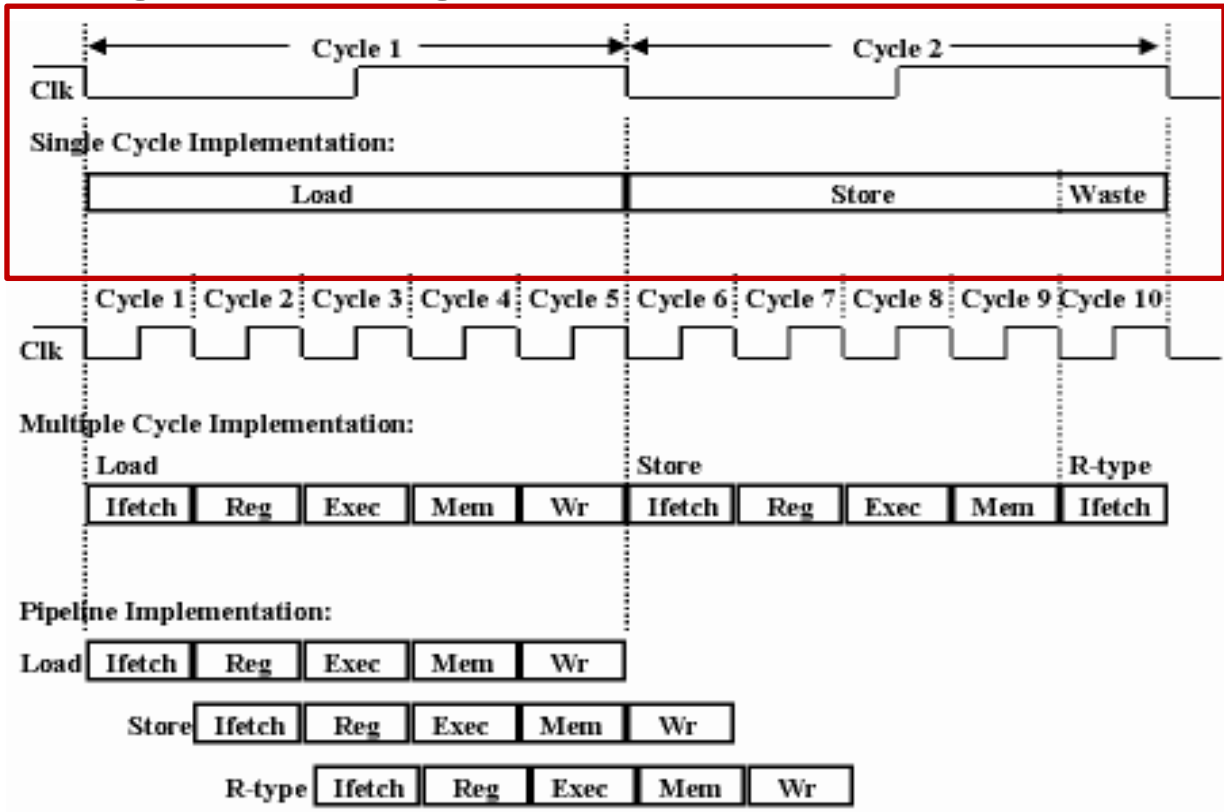
ARM Microarchitetture a ciclo singolo

Saranno prese in considerazione tre diverse microarchitetture ARM, le quali differiscono fra loro principalmente per il modo in cui gli elementi di stato sono interconnessi.

Ciclo singolo: l'intera istruzione è eseguita in un singolo ciclo.

Vantaggi:
- semplice da comprendere;
- unità di controllo molto semplice;
- non richiede stati non architetturali.

Svantaggi:
- tempo pari a quello dell'istruzione più lenta;
- memoria dati e memoria istruzioni separate;



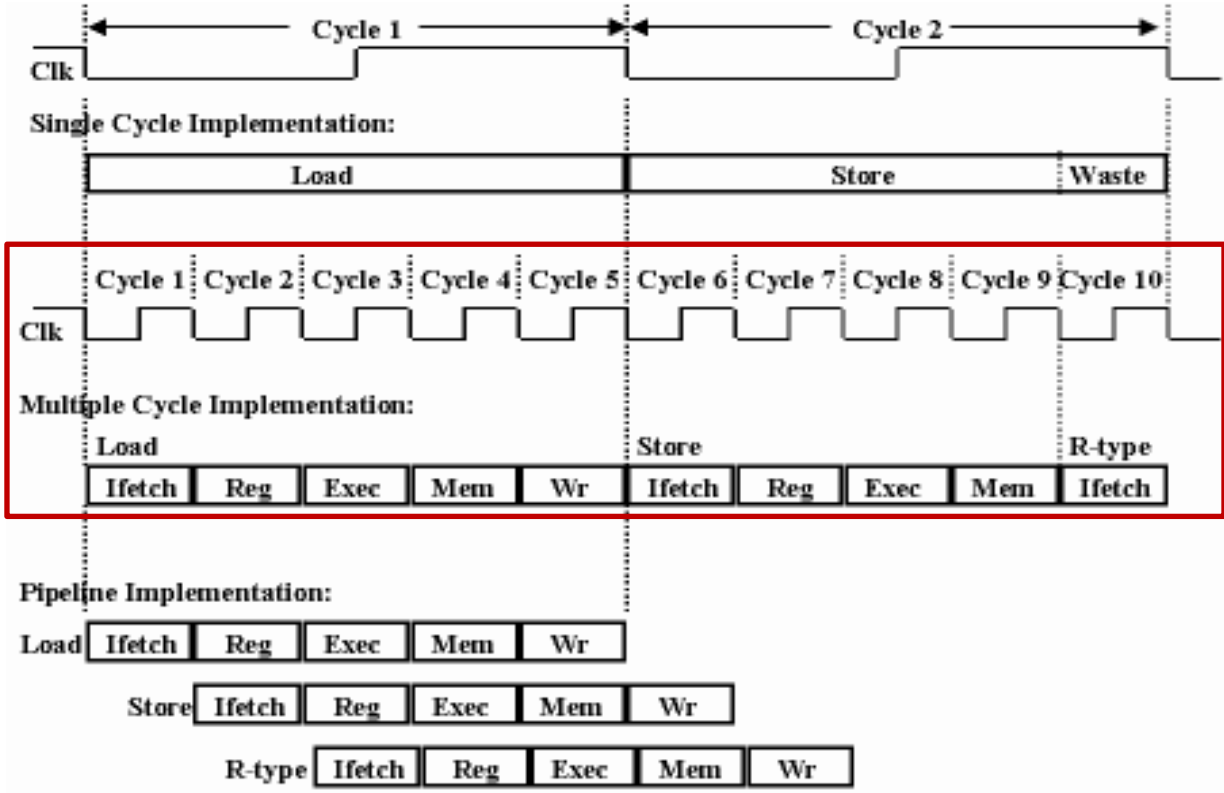
Microarchitetture a ciclo multiplo

Saranno prese in considerazione tre diverse microarchitetture ARM, le quali differiscono fra loro principalmente per il modo in cui gli elementi di stato sono interconnessi.

Ciclo multiplo: esegue una istruzione in più cicli brevi.

Vantaggi:
- riuso dei componenti;
- durata variabile delle istruzioni;
- non richiede la separazione delle memorie.

Svantaggi:
- richiede stati non architetturali;
- esegue una istruzione per volta.



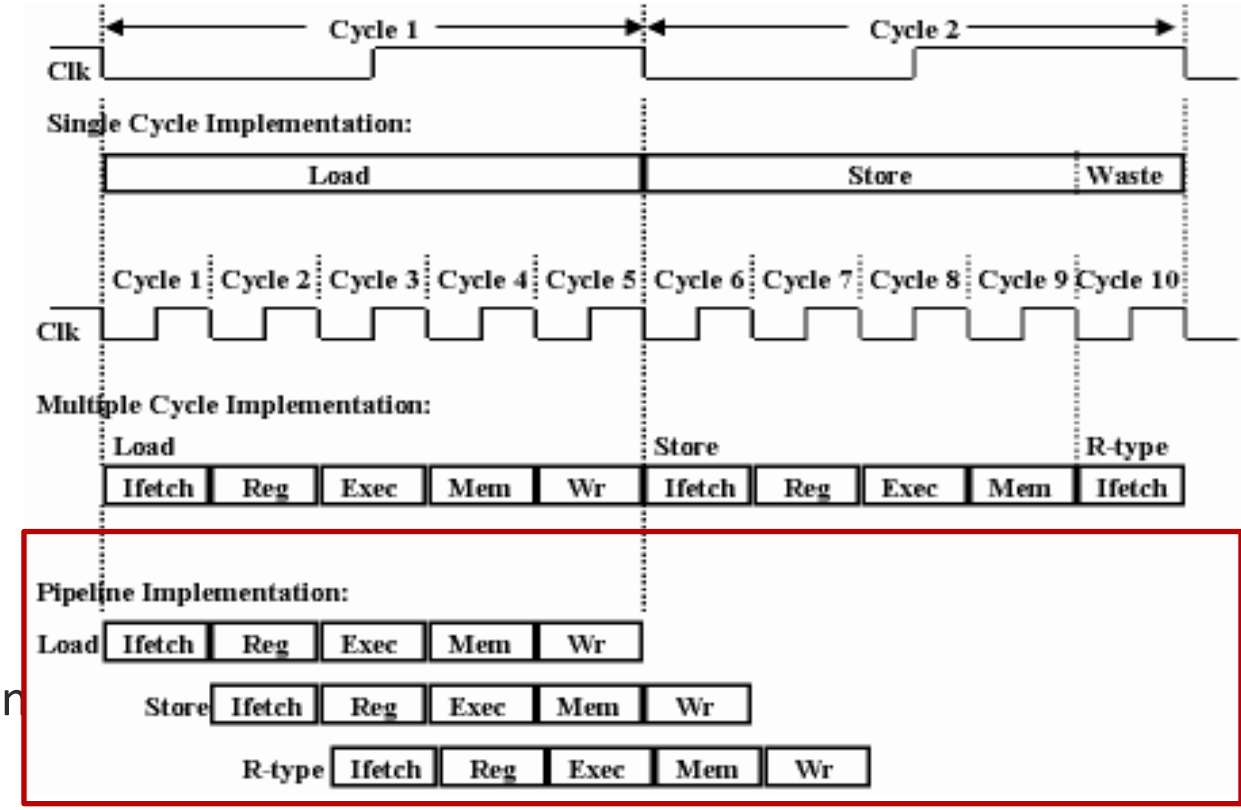
Microarchitetture con pipeline

Saranno prese in considerazione tre diverse microarchitetture ARM, le quali differiscono fra loro principalmente per il modo in cui gli elementi di stato sono interconnessi.

Pipelining: si applica al processore a ciclo singolo e ne migliora le performance.

Vantaggi:
- esegue più istruzioni contemporaneamente;
- si può accedere a dati e registri contemporaneamente.

Svantaggi:
- logica di controllo più complessa.
- richiede registri di pipelin



Misura delle prestazioni

Ci sono molti modi per misurare le performance di un processore.

Una misura affidabile consiste nel valutare le prestazioni di tempo rispetto all'esecuzione di un insieme fissato di programmi, che prende il nome di benchmark.

CINT2006 (Integer Component of SPEC CPU2006):

Benchmark	Language	Application Area	Brief Description
400.perlbench	C	Programming Language	Derived from Perl V5.8.7. The workload includes SpamAssassin, MHonArc (an email indexer), and specdiff (SPEC's tool that checks benchmark outputs).
401.bzip2	C	Compression	Julian Seward's bzip2 version 1.0.3, modified to do most work in memory, rather than doing I/O.
403.gcc	C	C Compiler	Based on gcc Version 3.2, generates code for Opteron.
429.mcf	C	Combinatorial Optimization	Vehicle scheduling. Uses a network simplex algorithm (which is also used in commercial products) to schedule public transport.
445.gobmk	C	Artificial Intelligence: Go	Plays the game of Go, a simply described but deeply complex game.
456.hmmer	C	Search Gene Sequence	Protein sequence analysis using profile hidden Markov models (profile HMMs)
458.sjeng	C	Artificial Intelligence: chess	A highly-ranked chess program that also plays several chess variants.
462.libquantum	C	Physics / Quantum Computing	Simulates a quantum computer, running Shor's polynomial-time factorization algorithm.
464.h264ref	C	Video Compression	A reference implementation of H.264/AVC, encodes a videostream using 2 parameter sets. The H.264/AVC standard is expected to replace MPEG2
471.omnetpp	C++	Discrete Event Simulation	Uses the OMNet++ discrete event simulator to model a large Ethernet campus network.
473.astar	C++	Path-finding Algorithms	Pathfinding library for 2D maps, including the well known A* algorithm.
483.xalancbmk	C++	XML Processing	A modified version of Xalan-C++, which transforms XML documents to other document types.

Misura delle prestazioni

Ci sono molti modi per misurare le performance di un processore.

Il tempo di esecuzione è calcolato come:

$$Tempo\ di\ esecuzione = (\#istruzioni)\left(\frac{cicli}{istruzione}\right)\left(\frac{secondi}{ciclo}\right)$$

CPI: Cycles/instruction

clock period: seconds/cycle

IPC: instructions/cycle = IPC

Misura delle prestazioni

Ci sono molti modi per misurare le performance di un processore.

Il tempo di esecuzione è calcolato come:

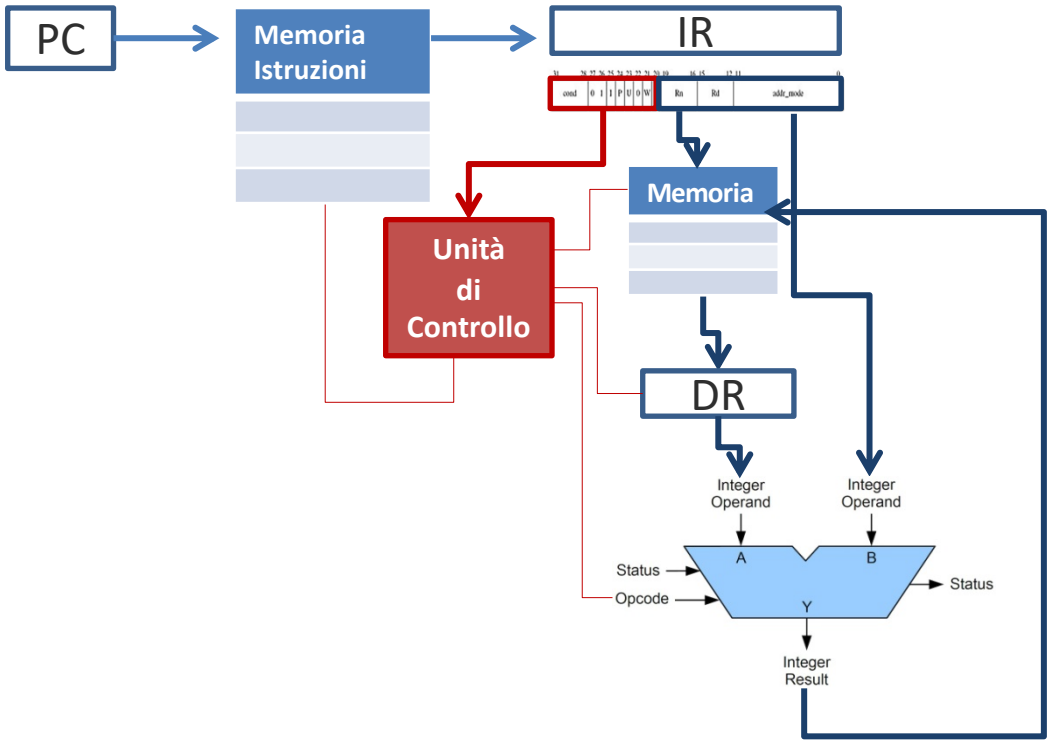
$$Tempo\ di\ esecuzione = (\#istruzioni)\left(\frac{cicli}{istruzione}\right)\left(\frac{secondi}{ciclo}\right)$$

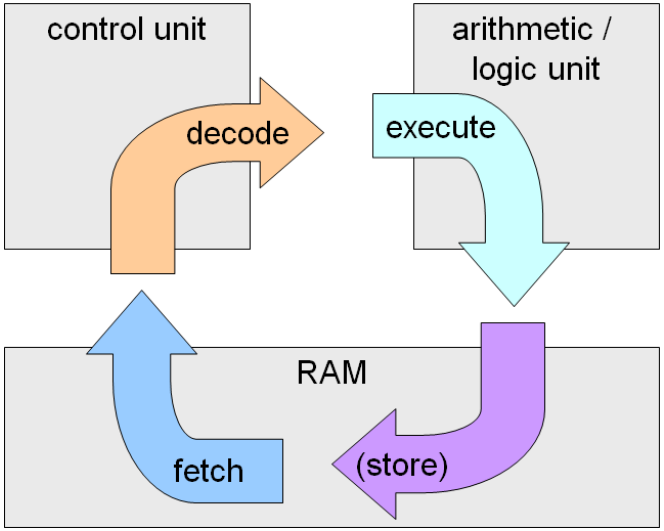
CPI: Cycles/instruction
clock period: seconds/cycle
IPC: instructions/cycle = IPC

La frequenza di clock corrisponde all'inverso del clock period

Microarchitetture: schema generale

Una istruzione ha un ciclo di vita che consta di quattro fasi principali





Progettazione

Una microarchitettura consta di due componenti che interagiscono fra loro:

- il **datapath:**
  - determina il flusso dei dati durante l'esecuzione di una istruzione
  - opera su parole dati e contiene strutture quali le memorie, i registri, l'ALU, e i multiplexer
  - considereremo un datapath a 32 bit.
- l'**unità di controllo:**
  - riceve l'istruzione corrente dal datapath ed "istruisce" il datapath su come eseguire tale istruzione.
  - Produce i valori di selezione dei multilpexer, i segnali di abilitazione alla scrittura dei registri e della memoria.

Procederemo in modo incrementale: Elementi di stato (program counter, registri, e registro di stato). Logica combinatoria fra gli elementi di stato. Gestione della memoria.

Architectural State Elements

Determines everything about a processor:

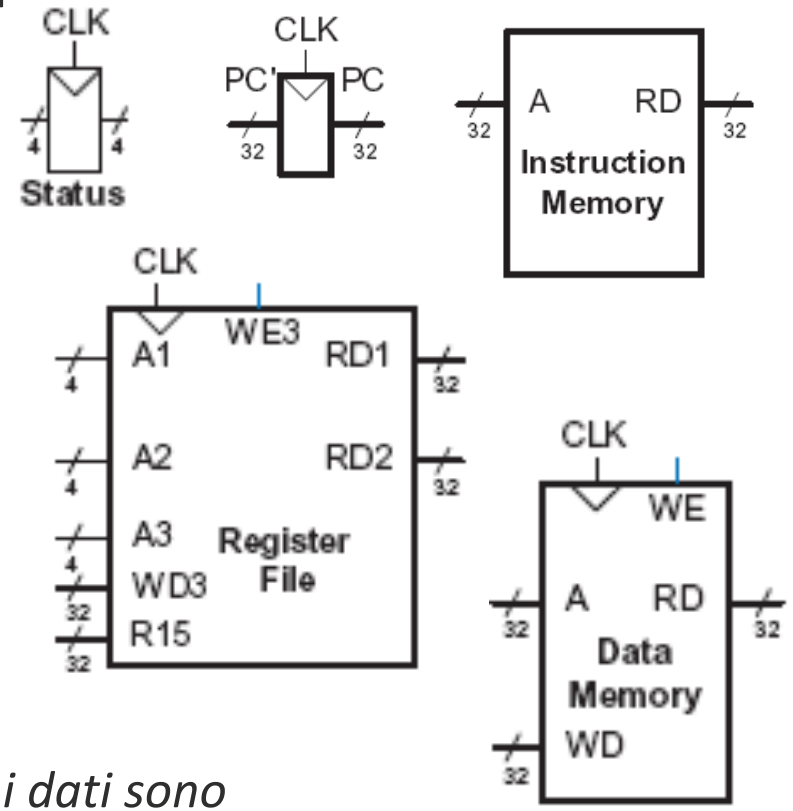
- Architectural state:
  - 16 registers (including PC)
  - Status register
- Memory

Progettazione

Il miglior modo di procedere nel processo di progettazione consiste nel considerare prima gli elementi di stato (program counter, registri, registro di stato) e aggiungere successivamente la logica combinatoria.

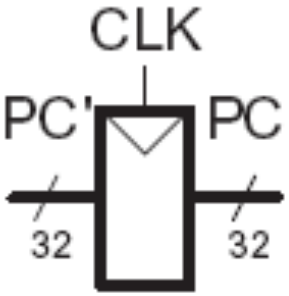
Suddivideremo la memoria in cinque elementi di stato:

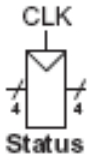
- il program counter
- i registri (register file)
- il registro di stato (status register)
- la memoria istruzioni (instruction memory)
- la memoria dati (data memory).

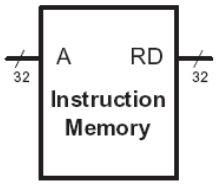


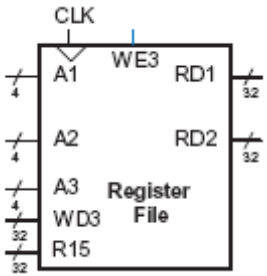
La memoria per le istruzioni e quella per i dati sono separate per il solo scopo di facilitare la comprensione.

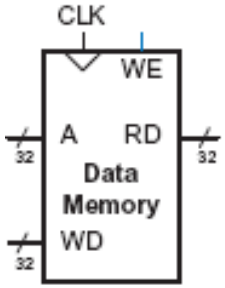
Program counter





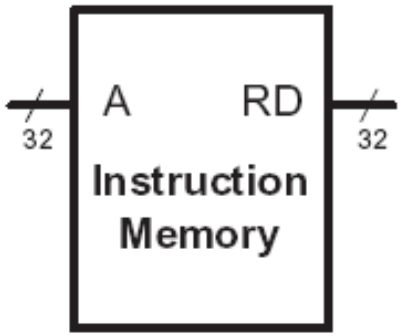






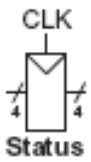
Sebbene il PC sia logicamente parte del file registro, esso viene letto e scritto ad ogni ciclo indipendentemente dagli altri registri ed è quindi implementato come un registro autonomo 32-bit. La sua uscita, PC, indica l'indirizzo dell'istruzione corrente. Il suo ingresso, PC', indica l'indirizzo della successiva istruzione.

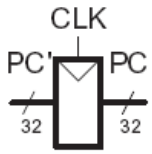
Istruction Memory

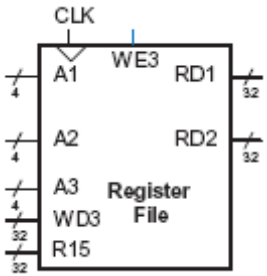


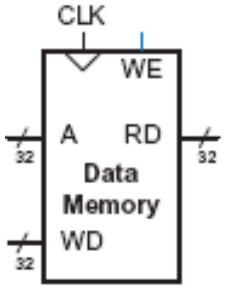
essa ha una singola porta di sola lettura (*semplificazione*).

A indica l'indirizzo di una istruzione che verrà riportata (*letta*) nel registro **RD**.

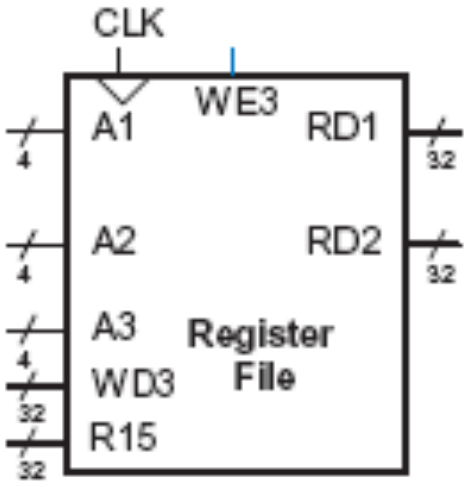


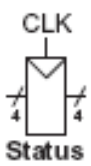


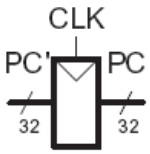


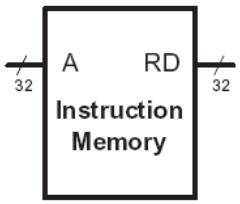


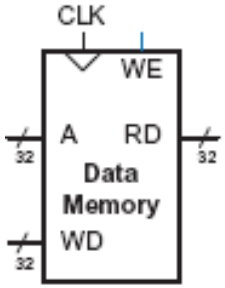
File register











consiste di 16 registri (**R0,…,R15**) a 32 bit

Dal punto di vista logico i registri non sono equivalenti, svolgono al livello architetturale funzioni diverse. In particolare:

R13: stack pointer

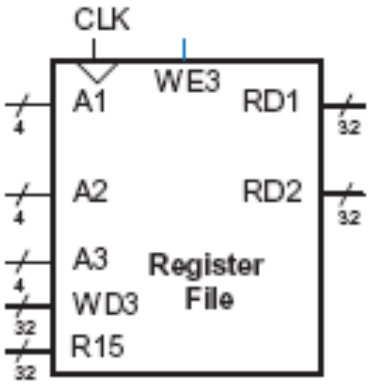
R14: link register

R15: proveniente dal program counter

File register

Esso ha due porte per la lettura **A1** e **A2** ed una per la scrittura **A3**.

Ciascuna porta di lettura ha un input di 4 bit (2^4=16), che specificano uno dei registri come operando, che viene letto nei registri **RD1** o **RD2**, rispettivamente.

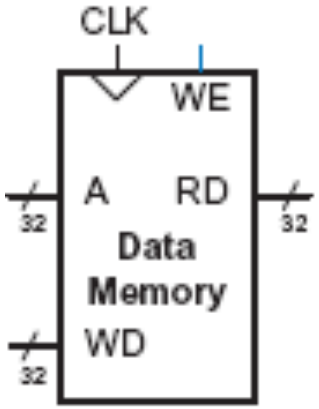


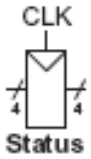
La porta di scrittura ha:

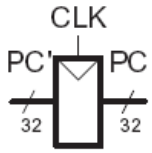
- un indirizzo di 4 bit **A3** (dove scrivere);
- un elemento di 32 bit **WD3** (cosa scrivere);
- un segnale di Write Enable (**WE3**).

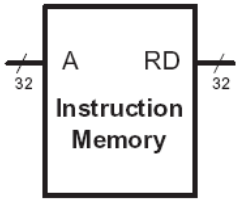
Se **WE** è 1, il dato **WD3** è scritto nel registro specificato da **A3**.

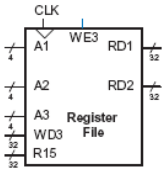
Data Memory





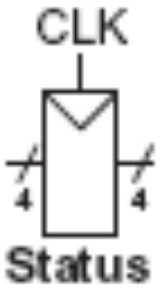




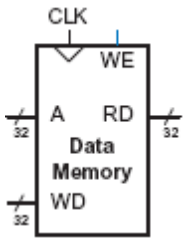


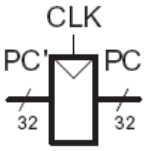
Il **Data Memory**: ha una singola porta di lettura/scrittura.
Quando il segnale **WE** (Write Enable) è attivo, essa scrive il dato **WD** nella cella puntata dall'indirizzo **A** durante il fronte alto del clock.
Quando il segnale **WE** (Write Enable) è 0, essa legge i dati nella cella puntata dall'indirizzo **A** durante il fronte alto del clock e li pone nel registro **RD**.

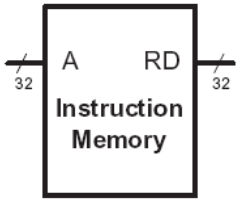
Status

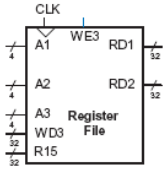


Memorizza i flags **N**,**Z**,**C**,**V** fornite dall'ALU al fine di eseguire istruzioni condizionate









Flag	Name	Description
N	**N**egative	Instruction result is negative
Z	**Z**ero	Instruction results in zero
C	**C**arry	Instruction causes an unsigned carry out
V	o**V**erflow	Instruction causes an overflow

ARM Processor Istruzioni di base

Al fine di facilitare la comprensione dell'architettura di un processore ARM, considereremo un limitato **set** di istruzioni:

- **Istruzioni di Data-processing:**
  - `ADD, SUB, AND, ORR`
  - Con registri e modalità di indirizzamento diretto, ma senza shifts
- **Istruzioni di memoria:**
  - `LDR, STR`
  - with **positive immediate offset**
- **Istruzioni di salto:**
  - `B`

Reading Memory

Vi sono anche altre segnature di LDR, ad esempio:

```
LDR R0, [R1, R2]
```

R0 ← mem32[R1 + R2]

```
LDR R0, [R1, R2, LSL #2]
```

R0 ← mem32[R1 + (R2*4)]

Reading Memory

Una istruzione che scrive in memoria è detta ***store***

- **Mnemonico:** `STR` (*store register*)
- Semantica speculare a LDR:

```
STR R0, [R1, #12]
```

mem32[R1 + 12] ← R0

- Segnature ananoghe

```
STR R1, [R0, R2]
```

mem32[R0 + R2] ← R1

```
STR R0, [R1, R2, LSL #1]
```

mem32[R1 + (R2*2)] ← R0

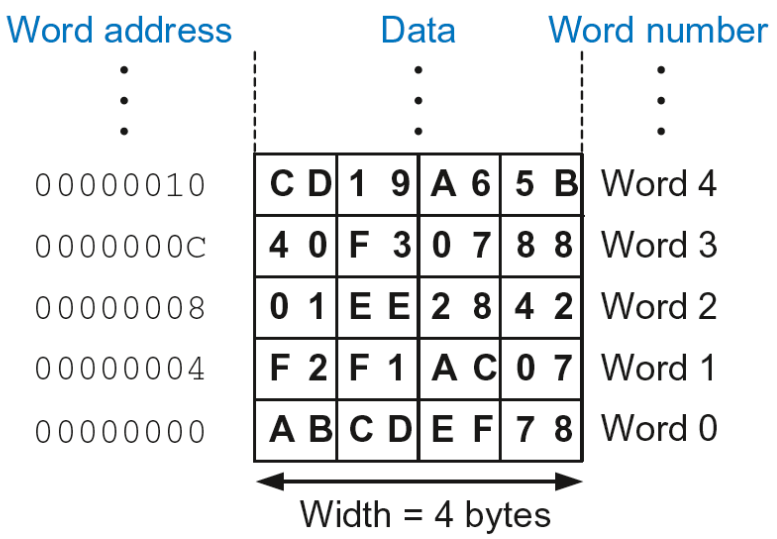
Writing Memory

Esempio: Memorizza il valore di R7 nella parola di memoria 21.

Indirizzo di memoria= 4 x 21 = 84 = 0x54

```
MOV R5, #0
STR R7, [R5, #0x54]
```

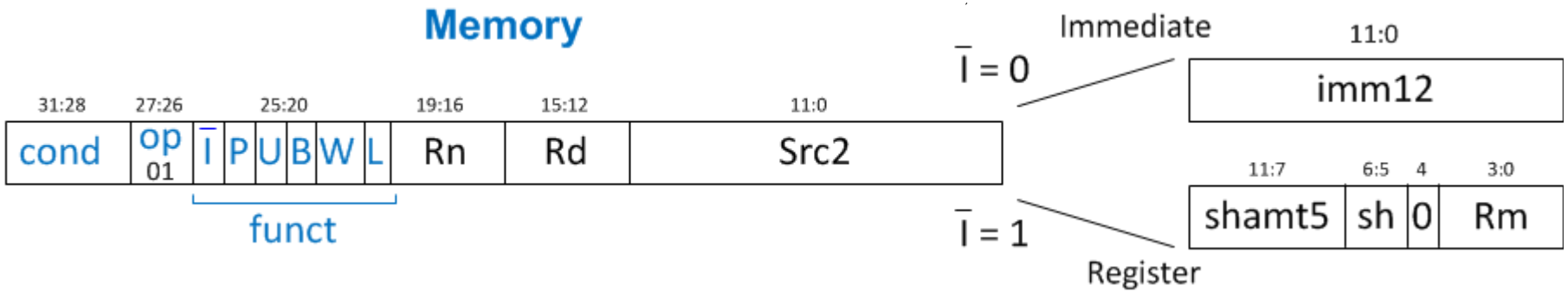
L'offset può essere scritto in decimale o esadecimale



Memory Instruction Format

LDR, STR, LDRB, STRB

- ***op*** = 01_2
- ***Rn*** = registro *base address*
- ***Rd*** = destinazione (load), sorgente (store)
- ***Src2*** = offset: registro (*shiftato* opzionalmente) o immediate
- ***funct*** = 6 bit di controllo



Indicizzazione

ARM fornisce tre tipi di indicizzazione:

Offset:
L'indirizzo è calcolato sommando o sottraendo un offset al contenuto del registro di base.

Preindex
L'indirizzo viene calcolato come nel caso dell'offset e viene scritto nel registro di base;

Postindex
L'indirizzo corrisponde al contenuto del registro di base;
L'offset viene aggiunto o sottratto al contenuto del registro di base e riscritto nel registro di base

Table 6.4 ARM indexing modes

Mode	ARM Assembly	Address	Base Register
Offset	`LDR R0, [R1, R2]`	R1 + R2	Unchanged
Pre-index	`LDR R0, [R1, R2]!`	R1 + R2	R1 = R1 + R2
Post-index	`LDR R0, [R1], R2`	R1	R1 = R1 + R2

Indicizzazione

ARM fornisce tre tipi di indicizzazione:

Offset:
L'indirizzo è calcolato sommando o sottraendo un offset al contenuto del registro di base.

Preindex
L'indirizzo viene calcolato come nel caso dell'offset e viene scritto nel registro di base;

Postindex
L'indirizzo corrisponde al contenuto del registro di base;
L'offset viene aggiunto o sottratto al contenuto del registro di base e riscritto nel registro di base

Esempi

Offset:
```
LDR R1, [R2, #4]          ; R1 = mem[R2+4]
```

Preindex:
```
LDR R3, [R5, #16]!        ; R3 = mem[R5+16]
                          ; R5 = R5 + 16
```

Postindex:
```
LDR R8, [R1], #8          ; R8 = mem[R1]
                          ; R1 = R1 + 8
```

Memory Format *funct* Encodings

Type of Operation

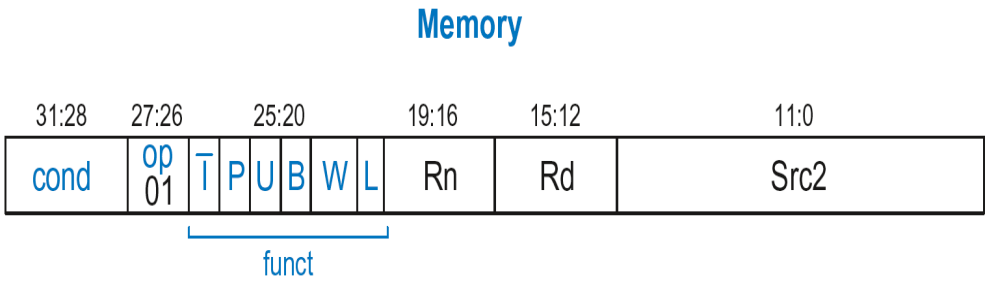
L	*B*	Instruction
0	0	`STR`
0	1	`STRB`
1	0	`LDR`
1	1	`LDRB`

Indexing Mode

P	*W*	Indexing Mode
0	1	Not supported
0	0	Postindex
1	0	Offset
1	1	Preindex

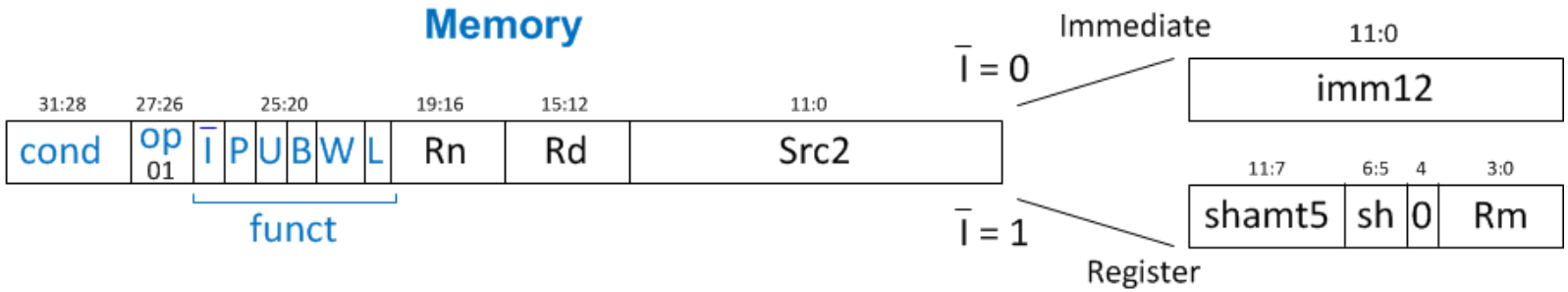
Add/Subtract Immediate/Register Offset

Value	$\overline{I}$	*U*
0	**Immediate** offset in *Src2*	**Subtract** offset from base
1	**Register** offset in *Src2*	**Add** offset to base



Memory Instruction Format

Sh codifica il tipo di shift (i.e., >>, <<, >>>, ROR)



Shift Type	*sh*
LSL	00_2
LSR	01_2
ASR	10_2
ROR	11_2

Memory Instr. with Immediate *Src2*

```
STR R11, [R5], #-26
```

- **Operation:** mem[R5] <= R11; R5 = R5 - 26
- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 01_2 (1) for memory instruction
- ***funct*** = $0\bar{0}000000_2$ (0)
- ***I*** = 0 (immediate offset), ***P*** = 0 (postindex),
- ***U*** = 0 (subtract), ***B*** = 0 (store word), ***W*** = 0 (postindex),
- ***L*** = 0 (store)
- ***Rd*** = 11, ***Rn*** = 5, ***imm12*** = 26

Field Values

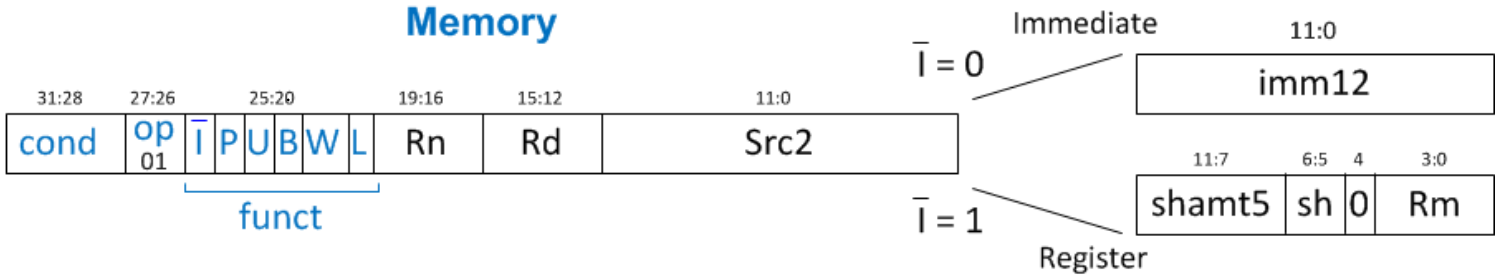
31:28	27:26	25:20	19:16	15:12	11:0
1110_2	01_2	0000000_2	5	11	26
cond	op	$\bar{I}$PUBWL	Rn	Rd	imm12
1110	01	000000	0101	1011	0000 0001 1010
E		4 0	5	B	0 1 A

Memory Instr. with Register *Src2*

```
LDR R3, [R4, R5]
```

- **Operation:** R3 <= mem[R4 + R5]
- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 01_2 (1) for memory instruction
- ***funct*** = $1\bar{1}1001_2$ (57)
- ***I*** = 1 (register offset), ***P*** = 1 (offset indexing),
- ***U*** = 1 (add), ***B*** = 0 (load **word**), ***W*** = 0 (offset indexing),
- ***L*** = 1 (load)
- ***Rd*** = 3, ***Rn*** = 4, ***Rm*** = 5 (***shamt5*** = 0, ***sh*** = 00)

1110 01 111001 0100 0011 00000 00 0 0101 = **0xE7943005**

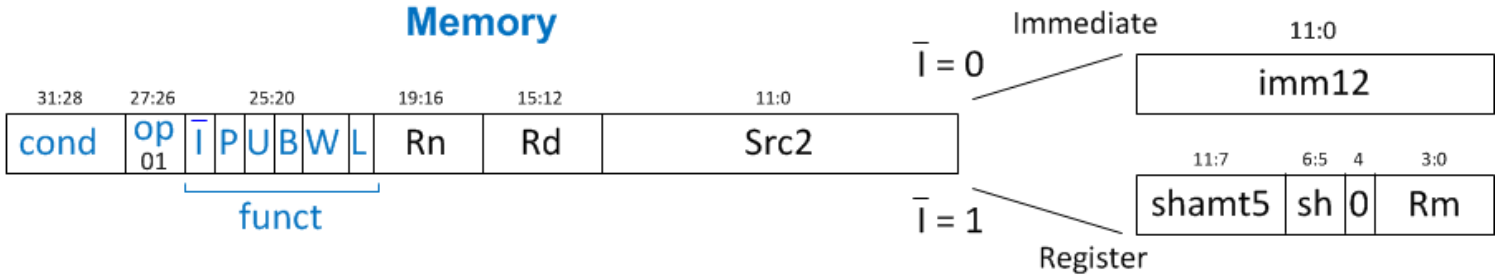


Memory Instr. with Scaled Reg. *Src2*

```
STR R9, [R1, R3, LSL #2]
```

- **Operation:** mem[R1 + (R3 << 2)] <= R9
- ***cond*** = 1110_2 (14) for unconditional execution
- ***op*** = 01_2 (1) for memory instruction
- ***funct*** = 111000_2 (0)
- ***I*** = 1 (register offset), ***P*** = 1 (offset indexing),
- ***U*** = 1 (add), ***B*** = 0 (store **word**), ***W*** = 0 (offset indexing),
- ***L*** = 0 (store)
- ***Rd*** = 9, ***Rn*** = 1, ***Rm*** = 3, ***shamt*** = 2, ***sh*** = 00_2 (LSL)

1110 01 111000 0001 1001 00010 00 0 0011 = **0xE7819103**



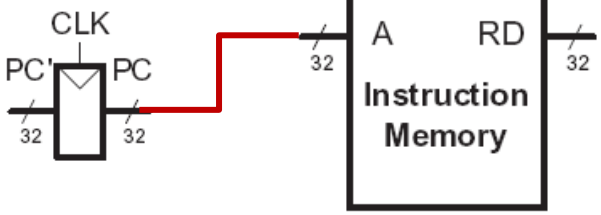
Datapath LDR (fetch)

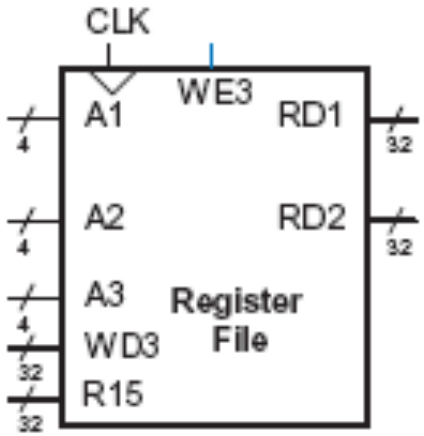
`LDR Rd, [Rn, imm12]`

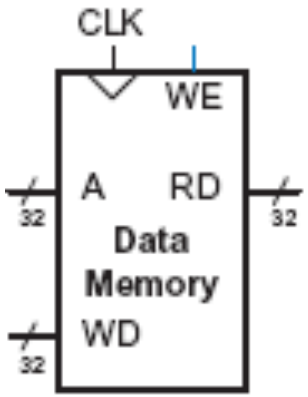
Il **PC** contiene l'indirizzo dell'istruzione da eseguire.

Il primo passo è quello di leggere questa istruzione dalla memoria istruzioni, per cui il PC viene collegato all'indirizzo di ingresso della memoria di istruzioni.

L'istruzione a 32 bit viene letta ed è rappresentata dall'etichetta **Instr**.



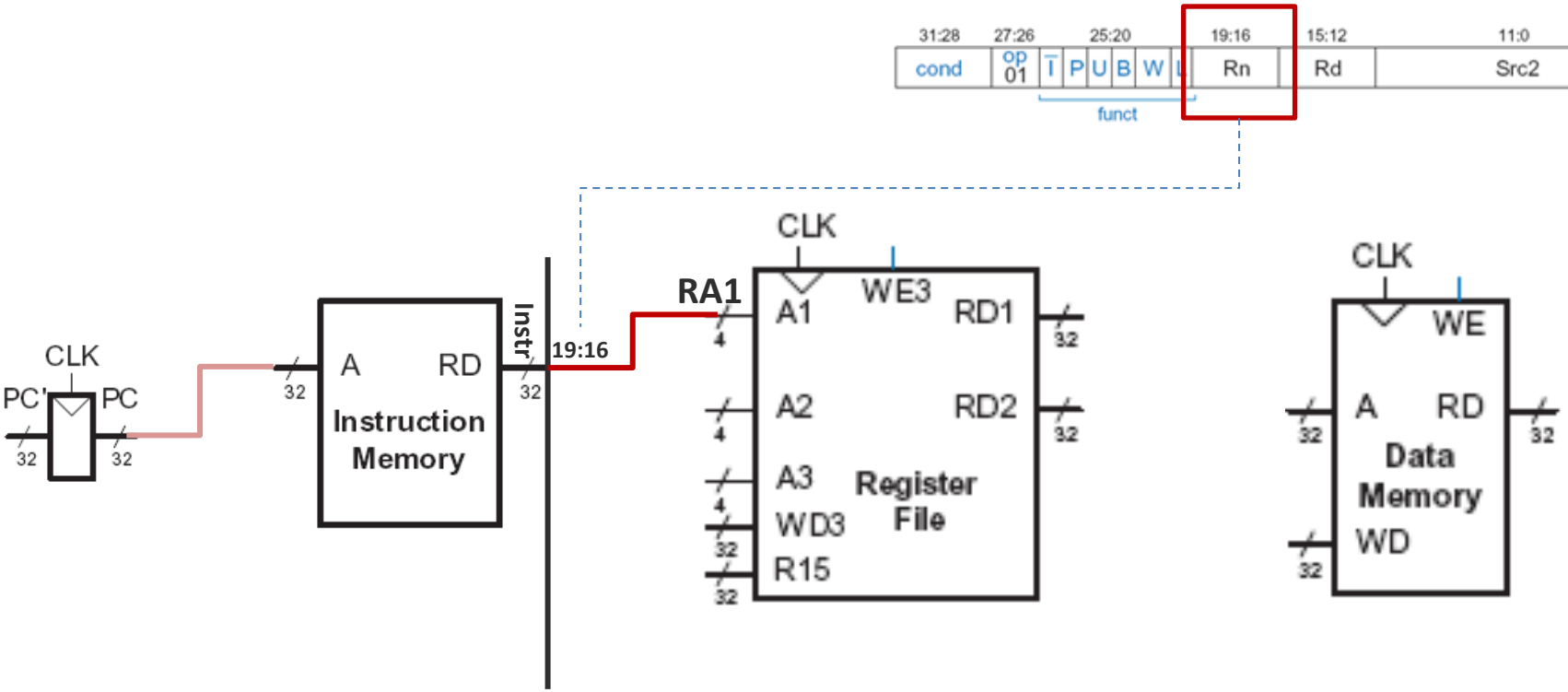




Datapath LDR

Il passo successivo è quello di leggere il registro contenente l'indirizzo di base. (L'indirizzo di) questo registro è specificato nel campo **Rn** dell'istruzione, **$Instr_{19:16}$**

Questi bit vengono collegati all'ingresso indirizzo di una delle porte del file register (**A1**). Il register file *legge* il valore di registro in **RD1**.

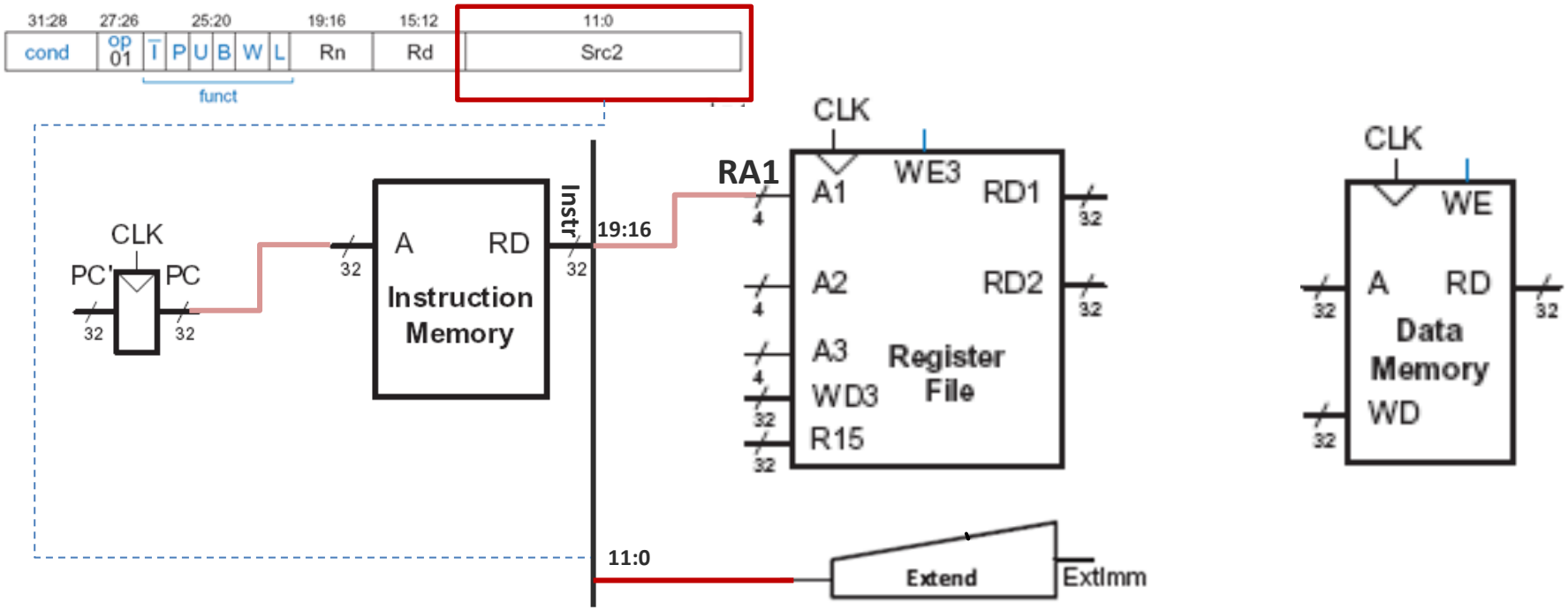


Datapath LDR

L'istruzione **LDR** richiede anche un **offset**, il quale è memorizzato nell'istruzione stessa e corrisponde ai bit $\mathbf{Instr}_{11:0}$.

L'offset è un valore senza segno, quindi deve essere esteso a 32 bit.

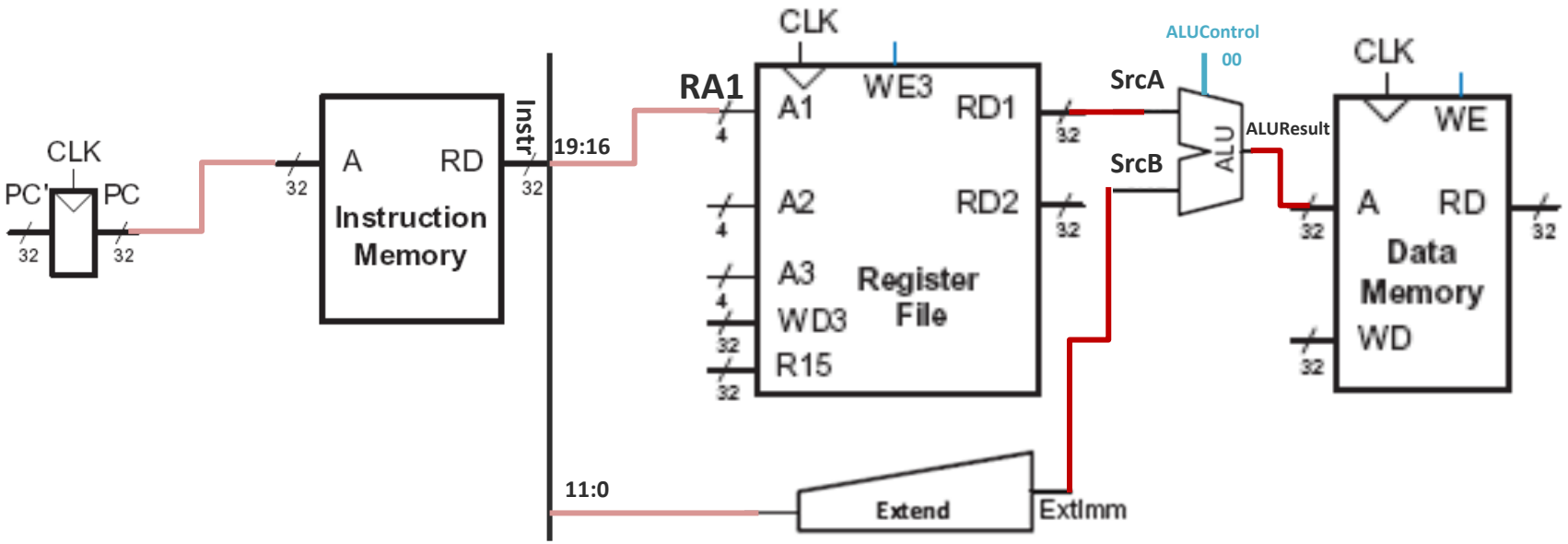
Il valore a 32 bit (**ExtImm**) è tale che $\mathbf{ExtImm}_{31:12} = 0$ e $\mathbf{ExtImm}_{11:0} = \mathbf{Instr}_{11:0}$.



Datapath LDR

Il processore deve aggiungere l'**indirizzo di base** all'offset per trovare l'indirizzo di memoria a cui leggere. La somma è effettuata per mezzo di una **ALU**.

La **ALU** riceve due operandi (**srcA** e **srcB**). **srcA** proviene dal register file, mentre e **srcB** in questo esempio proviene da **ExtImm**. Inoltre, il segnale a 2-bit **ALUControl** specifica l'operazione: una somma (indicata con 00), se il valore del bit U è uguale a 1, una sottrazione (indicata 01), viceversa.
La **ALU** genera un valore a 32 bit **ALUResult**, che viene inviato alla memoria dati come indirizzo di lettura.

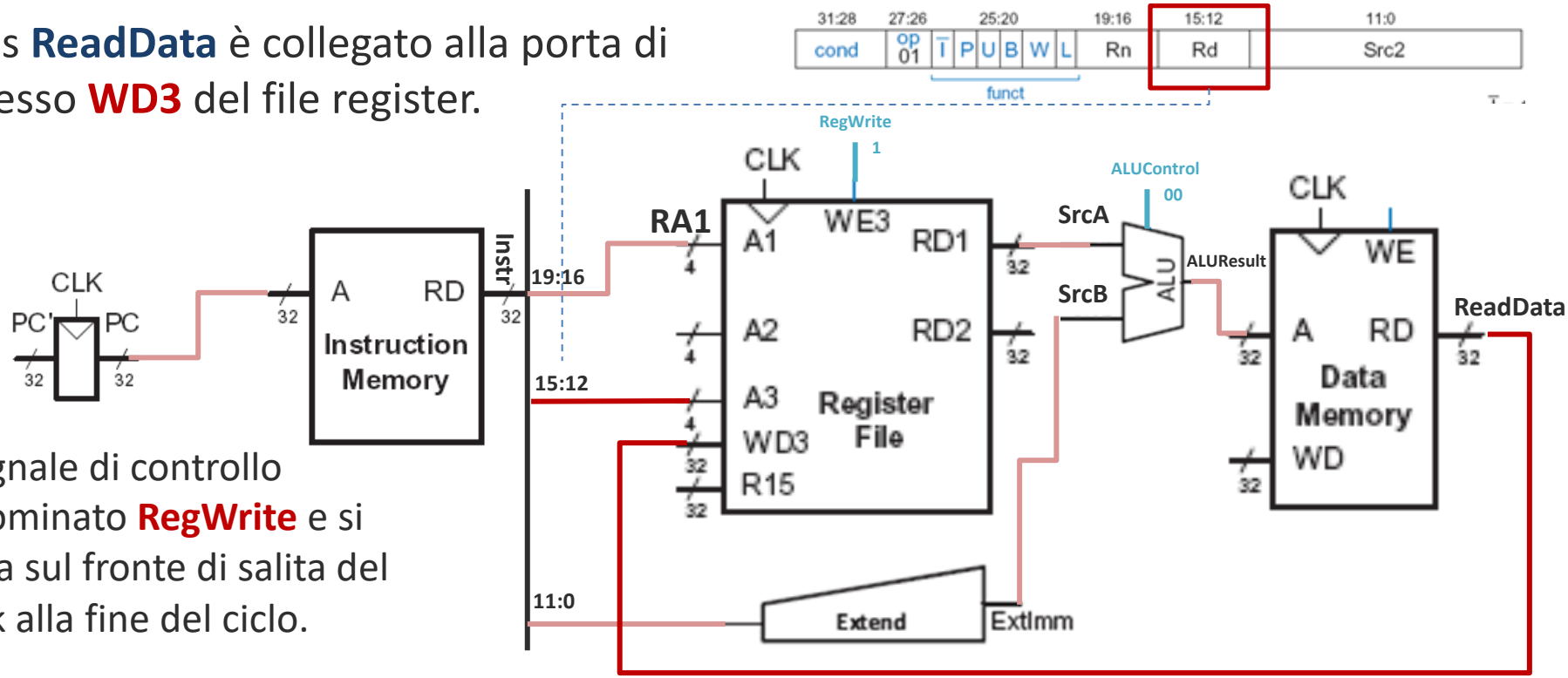


Datapath LDR

I dati vengono letti dalla memoria dati sul bus **ReadData** e poi sono scritti nel registro destinazione alla fine del ciclo.

il registro di destinazione per l'istruzione **LDR** è specificato nel campo **Rd**, **Instr$_{15:12}$**, che è collegato all'indirizzo di ingresso **A3**.

Il bus **ReadData** è collegato alla porta di ingresso **WD3** del file register.



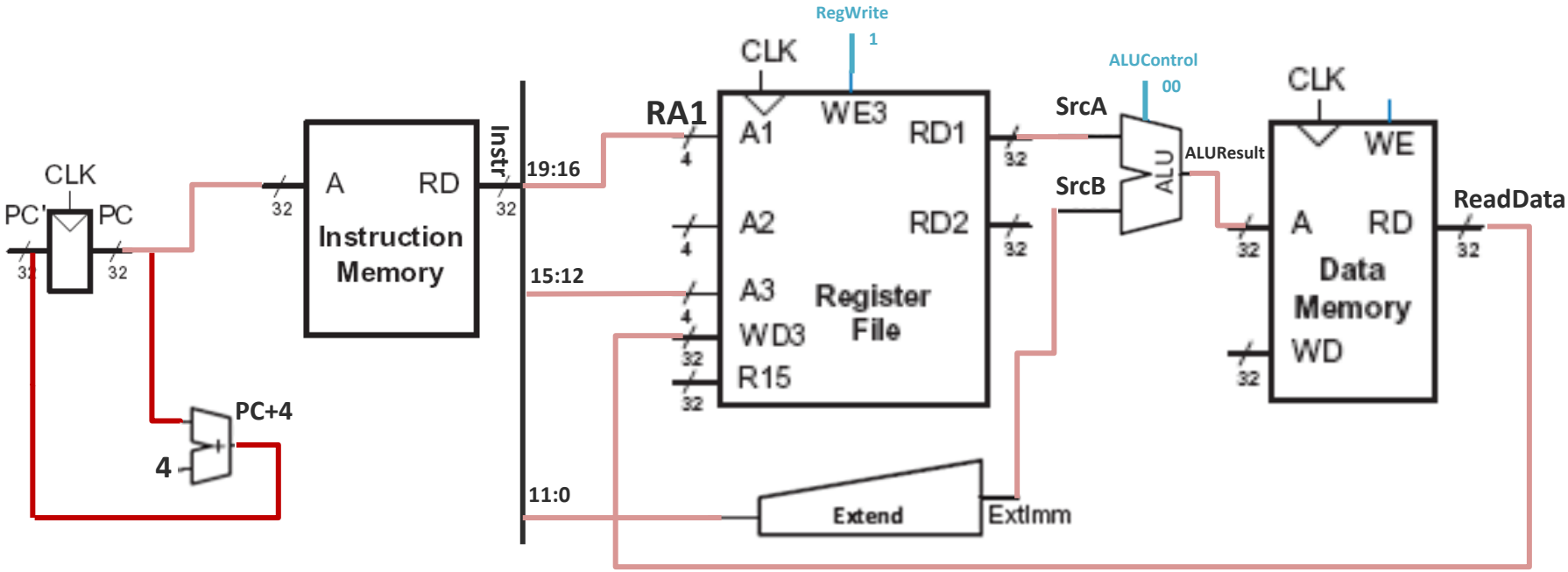
Il segnale di controllo denominato **RegWrite** e si attiva sul fronte di salita del clock alla fine del ciclo.

Datapath LDR

Contemporaneamente il processore deve calcolare l'indirizzo della successiva istruzione **PC '**.

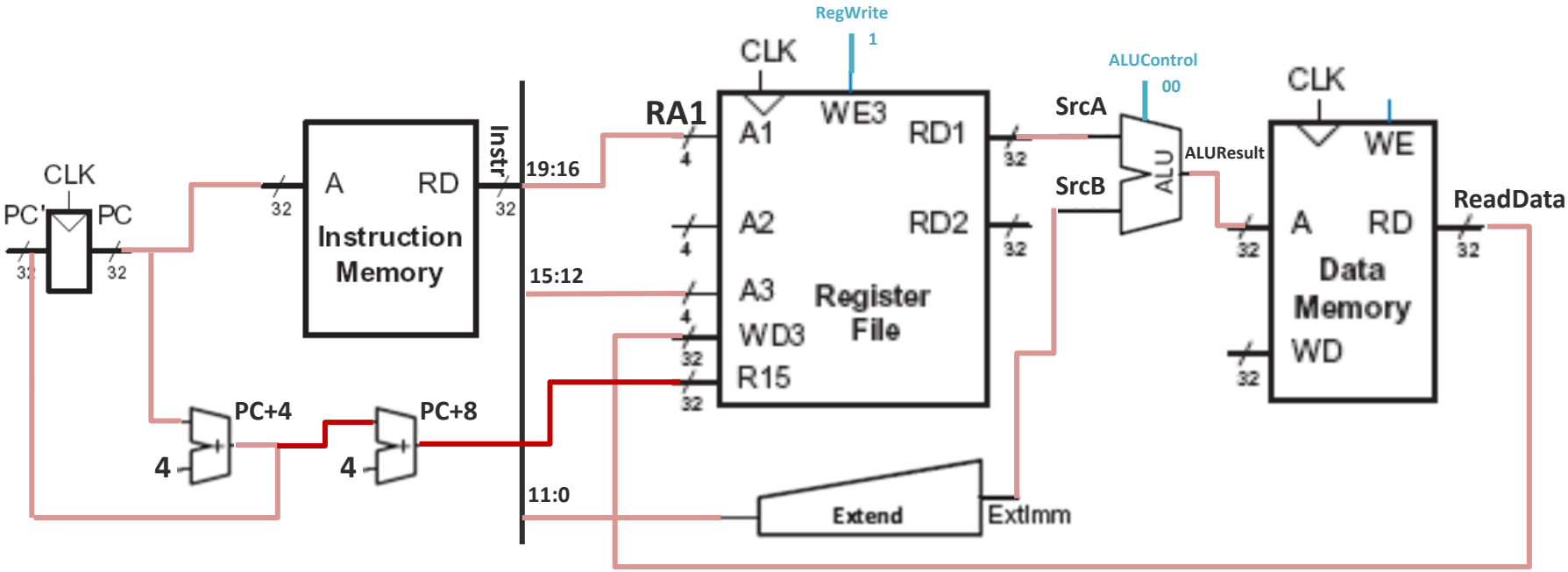
Le istruzioni sono a 32 bit (4 byte), quindi l'istruzione successiva è a **PC + 4**.

Si utilizza un **sommatore** per incrementare il **PC** di 4. Il nuovo indirizzo viene scritto nel contatore di programma sul successivo fronte di salita del clock.



Datapath LDR

Nelle architetture ARM il registro **R15** contiene il valore **PC+8**, per cui è necessario un ulteriore **sommatore** (+4), la cui uscita sia collegata all'ingresso di **R15**.



Datapath LDR

Infine, trattandosi di uno stored program paradigm, l'istruzione successiva potrebbe essere letta anche dalla memoria e quindi corrispondere al contenuto di **ReadData**. Un **multiplexer**, permette di selezionare fra:

- 0 – **PCPlus4**
- 1 – **ReadData**.

Il segnale di controllo associato al multiplexer è **PCSrc**.