Linguaggio Assembly

Architettura di riferimento

Formato istruzioni

Classi di istruzioni

Modalità di indirizzamento

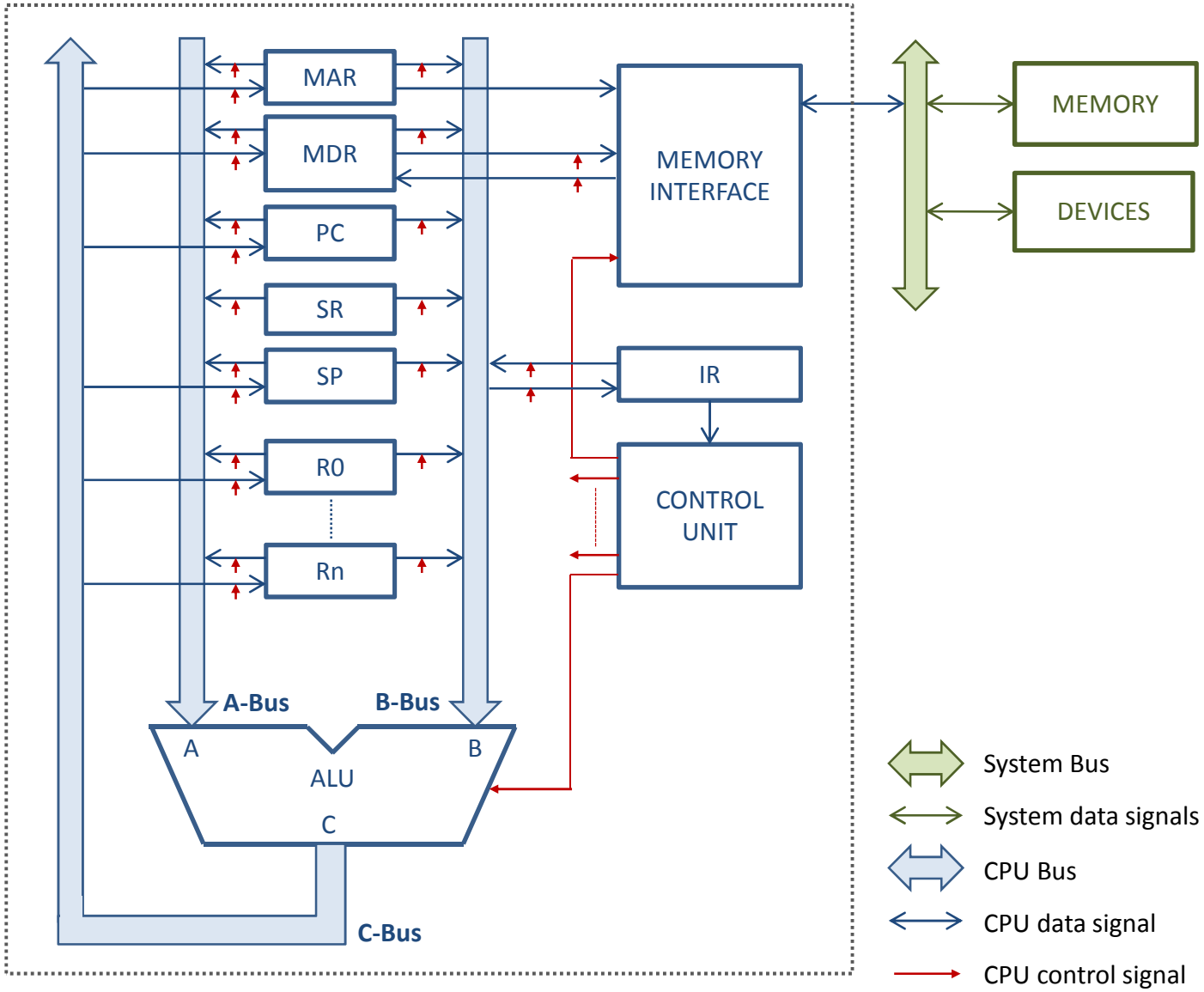
Direttive

Codice macchina

Modello di architettura di riferimento

- **Nel seguito faremo riferimento ad un modello di architettura a tre bus**
  - Il linguaggio descritto è tuttavia adatto anche ad architetture a due bus
  - Nel caso di architetture a un solo bus, il linguaggio richiede alcune semplificazioni
- **Ricordiamo che l'architettura di base è è costituita da**
  - Un insieme di registri "general purpose", o GPR
    - Utilizzati per contenere temporaneamente i dati cu cui il programma lavora
  - Alcuni registri speciali (SR, PC, SP, MAR, MDR, ...)
    - Mantengono memorizzate informazioni necessarie per l'esecuzione
  - Una unità di interfaccia verso la memoria centrale
    - Consente l'accesso ai dati e al codice del programma
  - Una unità aritmetico-logica, o ALU
    - Svolge le operazioni aritmetiche e logiche sui dati
  - Una struttura di interconnessione di queste unità basato su bus
    - Possono essere 1, 2 o 3, a seconda dell'architettura specifica
  - Una unità di controllo
    - Si occupa di interpretare il significato di un'istruzione assembly e di "configurare" e "coordinare" le altre unità in modo da eseguire l'operazione richiesta

Architettura di riferimento



Formato istruzioni

- **Il linguaggio assembly**
  - E' costituito da qualche decina di istruzioni
  - Le istruzioni hanno un formato testuale cosiddetto "simbolico", cioè leggibile
  - Le istruzioni svolgono operazioni molto semplici
  - Le istruzioni manipolano dati di tipo elementare, tipicamente "parole"

- **Codice macchina**
  - E' la traduzione la traduzione in binario del codice assembly
  - Non è leggibile da una parsona
  - Ogni istruzione assembly viene tradotta da una parola di codice macchina
  - Istruzioni assebly di una stessa classe hanno un formato macchina simile
  - La dimensione delle parole macchina può essere fissa o variabile

- **Assembler**
  - E' il progrmma che traduce, ovvero "assembla", il codice assembly in codice macchina
  - Si tratta di una traduzione pedissequa e non richiede alcuna "intelligenza"

Formato istruzioni

- **Una istruzione assembly è formata da**
  - Un'etichetta opzionale
    - Assegna un nome simbolico all'indirizzo di memoria in cui si troverà l'istruzione
  - Un codice operativo o "opcode"
    - Specifica l'operazione da svolgere
  - Zero, uno, due o tre operandi
    - Specificano i dati su cui operare
    - Si hanno operandi sorgente, operando destinazione e operando implicito

- **Il numero (massimo) di operandi di un dato linguaggio assebly è fissato**
  - Un operando
    - Macchine a pila e macchine con accumulatore
  - Due operandi
    - Macchine tipicamente CISC (Complex Instruction Set Architecture)
  - Tre operandi
    - Macchine tipicamente RISC (Reduced Instruction Set Architecture)

Formato istruzioni

- **Assembly ad un operando**
  - Al più un operando può essere specificato
  - Il secondo operando, quando richiesto, e/o la destinazione sono impliciti

```
NEG             ! acc <- -[acc]       Negates the value in the accumulator
LOAD X          ! acc <- [X]          Loads the value X in the accumulator
```

- **Assembly a due operandi**
  - Gli operandi indicano esplicitamente sorgenti e destinazione
  - In genere la destinazione coincide con uno dei due operandi sorgente

```
ADD R1, R2      ! R1 <- [R1]+[R2]     Adds R1 to R2 and store result in R1
LOAD R2,X       ! R2 <- [X]           Loads the value X in R2
```

- **Assembly a tre operandi**
  - Tutti gli operandi sorgente e destinazione sono indicati esplicitamente

```
ADD R1, R2, R3  ! R1 <- [R2]+[R3]     Adds R2 to R3 and store result in R1
LOAD R2,X       ! R2 <- [X]           Loads the value X in R2
```

Notazione

- **Nello scrivere codice assembly generico useremo la seguente notazione**

Simbolo	Significato
`Rn`	Registro general purpose numero n
`SP,PC,...`	Registro speciale
`acc`	Registro accumulatore
`X, Y, ...`	Locazioni di memoria
`Rn(k)`	Bit in posizione k del registro general purpose numero n
`Lm`	Etichetta, numerate progressivamente
`[ ]`	Valore di un registro o di una locazione di memoria
`<-`	Operazione di copia di un valore in una destinazione
`!`	Commento

- **Si noti la differenza tra le seguenti scritture**
  - `R1` Indica il registro R1
  - `[R1]` Indica il valore del registro R1
  - `X` Indica la locazione di memoria X, quindi un indirizzo
  - `[X]` Indica il valore della locazione di memoria all'indirizzo X

Operazioni di trasferimento

- **Trasferimento registro-registro**
  - Trasferimento dal registro sorgente Rs al registro destinazione Rd

```
MOVE Rd, Rs          ! Rd <- [Rs]
```

- **Trasferimento immediato-registro**
  - Il termine "immediato" indica una costante esplicitata nell'istruzione stessa
  - Trasferisce il valore immediato IMM nel registro destinazione Rd

```
MOVE Rd, IMM         ! Rd <- IMM
```

- **Trasferimento memoria-registro**
  - Copia il valore della locazione di memoria all'indirizzo X nel registro destinazione Rd

```
LOAD Rd, X           ! Rd <- [X]
```

- **Trasferimento registro-memoria**
  - Copia il valore del registro sorgente Rs nell locazione di memoria all'ndirizzo X

```
STORE X, Rs          ! X <- [Rs]
```

Operazioni aritmetiche

- **Operazione binaria registro-registro**
  - Esegue un'operazione tra i registri Rs ed Rd e memorizza il risultato in Rd

```
ADD Rd, Rs          ! Rd <- [Rd] + [Rs]
```

- **Operazione binaria immediato-registro**
  - Esegue un'operazione tra l'immediato IMM ed Rd e memorizza il risultato in Rd

```
SUB Rd, IMM         ! Rd <- [Rd] - IMM
```

- **Operazione unaria su registro**
  - Esegue un'operazione unaria sul registro Rn e memorizza il risultato in Rn stesso

```
NEG Rn              ! Rn <- -[Rn]
CLR Rn              ! Rn <- 0
```

- **Le principali e più comuni operazioni aritmetiche sono**
  - Operazioni binarie: ADD, SUB, MUL, DIV, MOD
  - Operazioni unarie: NEG, CLR

Operazioni aritmetiche

- **Nelle architetture di tipo CISC è possibile operare direttamente in memoria**
- **Operazione registro-memoria**
  - Se l'operando destinazione è un registro, si ha un solo accesso alla memoria

```
ADD Rn, X          ! Rn <- [Rn] + [X]
```

  - Se l'operando destinazione è una locazione di memoria si hanno due accessi

```
ADD X, Rn          ! X <- [Rn] + [X]
```

- **Operazione immediato-memoria**
  - L'operando destinazione è sempre la memoria, pertanto si possono avere due accessi

```
ADD X, IMM         ! X <- [X] + IMM
CLR X              ! X <- 0
```

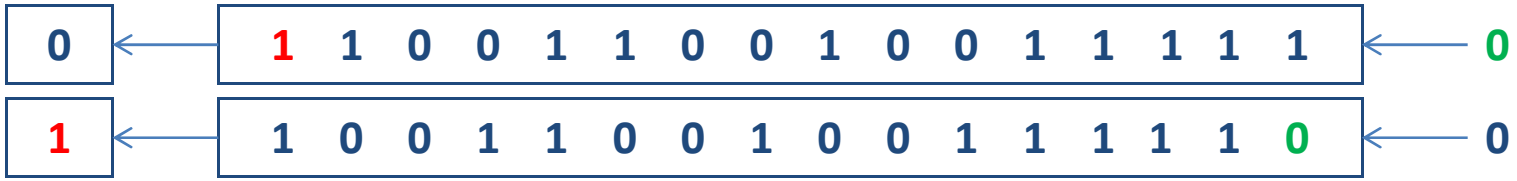
- **Più raramente è possibile eseguire operazioni tra due locazioni di memoria**
  - Tali istruzioni possono richiedere fino a tre accessi alla memoria

```
ADD X, Y           ! X <- [X] + [Y]
```

Operazioni di scorrimento

- **Le operazioni di scorrimento o shift**
  - Operano sui singoli bit dell'operando
  - Coinvolgono il bit SR(C), cioè il bit di carry del registro di stato
- **Esistono due tipi di scorrimento**
  - Logico: Tratta i bit come entità binarie, prive di un significato numerico
  - Aritmetico: Tratta i bit come parte di una parola avente un significato numerico
- **Shift logico a sinistra**

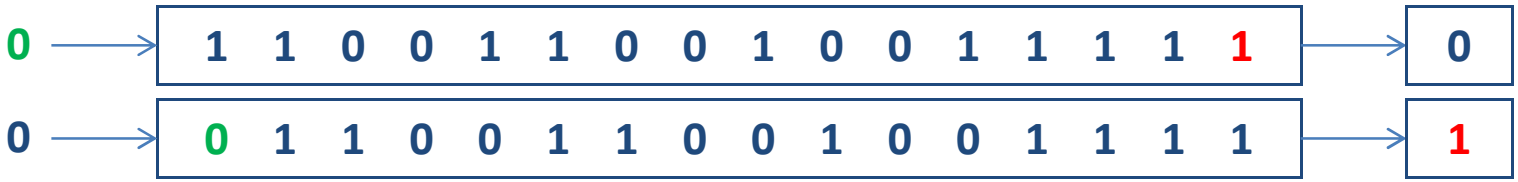
  - Esempio



Operazioni di scorrimento

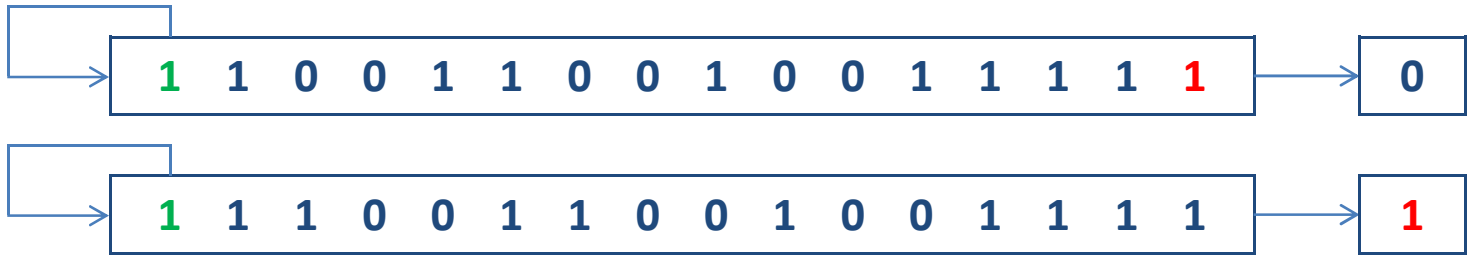
- **Shift logico a destra**

  - Esempio



- **Shift aritmetico a destra**

  - Esempio



Operazioni di scorrimento

- **Rotazione a sinistra**

  - Esempio

- **Rotazione a destra**

  - Esempio

Operazioni di scorrimento

- **Rotazione a sinistra con riporto**

  - Esempio

- **Rotazione a destra con riporto**

  - Esempio

Operazioni di scorrimento

- **Le istruzioni di scorrimento sono le seguenti**
  - Scorrimento a sinistra: LSL
  - Scorrimento a destra: LSR, ASR
  - Rotazione senza riporto: ROL, ROR
  - Rotazione con riporto: ROLC, RORC
- **Scorrimento di una sola posizione**
  - Si intende sottintesa l'entità dello scorrimento

```
LSL Rn              ! Rn <- [Rn(b-2:0)] & '0'
```

- **Scorrimento con immediato**
  - L'entità dello scorrimento è un valore costante

```
LSL Rn, IMM         ! Rn <- [Rn(b-1-IMM:0)] & IMM times '0'
```

- **Scorrimento con registro**
  - Talora è possibil specificare l'entità dello scorrimento mediante un registro

```
LSL Rn, Rs          ! Rn <- [Rn(b-1-[Rs]:0)] & [Rs] times '0'
```

Operazioni logiche

- **Ammettono gli stessi formati visti per le operazioni aritmetiche**
  - Tali operazioni considerano gli operandi come valori di verità o falsità
  - In generale
    - Il valore numerico 0 rappresenta il valore logico di falsità
    - Un valore numerico qualsiasi diverso da zero indica il valore di verità

- **Le principali operazioni logiche sono**
  - Operazioni binarie: AND, OR, XOR
  - Operazioni unarie: NOT

Operazioni di confronto

- **Le operazioni di confronto**
  - Non hanno un registro destinazione esplicito, ma modificano ibit del registro di stato
    - SR(N) Indica "negativo" o "minore di"
    - SR(Z) Indica "zero" o "uguale"
- **Confronto registro-zero**
  - Confronta il valore di un registro con il valore costante implicito zero

```
TEST Rn          ! SR(N) <- 1 if [Rn] < 0; 0 otherwise
                 ! SR(Z) <- 1 if [Rn] = 0; 0 otherwise
```

- **Confronto immediato-registro**
  - Confronta il valore di un registro con il valore immediato IMM

```
CMP Rn, IMM      ! SR(N) <- 1 if [Rn] < IMM; 0 otherwise
                 ! SR(Z) <- 1 if [Rn] = IMM; 0 otherwise
```

- **Confronto registro-registro**
  - Confronta il valore di un registro con il valore immediato IMM

```
CMP Rn, Rm       ! SR(N) <- 1 if [Rn] < [Rm]; 0 otherwise
                 ! SR(Z) <- 1 if [Rn] = [Rm]; 0 otherwise
```

Operazioni di confronto

- **Nella maggior parte delle architetture le operazioni aritmetiche e logiche**
  - Comportano l'esecuzione dell'operazione vera e propria
  - Eseguono automaticamente il confronto del risultato con il valore zero
    - Per risultato si intende sempre il valore memorizzato nell'operando destinazione
  - Impostano alcuni ulteriori bit del registro di stato, in particolare
    - SR(C) Indica che l'ultima operazione ha prodotto un riporto al bit più significativo
    - SR(V) Indica che l'ultima operazione (in complemento a due) ha prodotto un overflow
  - In questo modo è molto spesso possibile evitare istruzioni di confronto esplicite
- **Un'operazione aritmetico/logica dunque**
  - Modifica il registro destinazione o la locazione di memoria destinazione
  - Modifica i bit del registro di stato
  - Dovrebbe essere pertanto descritta come segue

```
ADD Rd, Rs      ! Rd <- [Rd] + [Rs]
                ! SR(N) <- 1 if [Rd] < 0; 0 otherwise
                ! SR(Z) <- 1 if [Rd] = 0; 0 otherwise
                ! SR(C) <- 1 if carry;    0 otherwise
                ! SR(V) <- 1 if overflow  0 otherwise
```

Operazioni di salto

- **Modificano il flusso di esecuzione del programma**
  - Modificano il contenuto del program counter
- **Salto incondizionato a etichetta (immediato)**
  - Sostituisce il program counter con il valore specificato nell'immediato

```
B    Lk            ! PC <- Lk
```

- **Salto incondizionato relativo**
  - Somma al program lo spiazzamento specificato nell'immediato

```
B    IMM(PC)       ! PC <- [PC] + IMM
```

- **Salto incondizionato a registro**
  - Sostituisce al program il valore specificato nell'immediato

```
B    Rs            ! PC <- Rs
```

Operazioni di salto

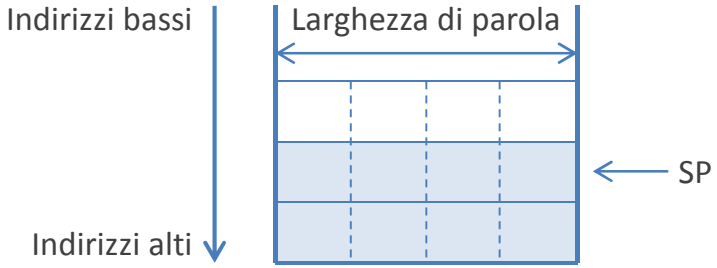
- **Salto condizionato**
  - Viene eseguito se e solo se una data condizione è verificata
  - Possiamo desrivere la sintassi generale indicando con
    - **`cond`** una generica condizione
    - **`dest`** la destinazione, in uno dei formati visti per i salti incondizionati

```
Bcond   dest      ! PC <- dest if cond = TRUE
```

- **Condizioni**
  - **`LT`** Less Than
  - **`LE`** Less than or Equal
  - **`GT`** Greater Than
  - **`GE`** Greater than or Equal
  - **`EQ`** Equal
  - **`NE`** Not Equal
  - **`C`** Carry
  - **`V`** Overflow

Operazioni di gestione dello stack

- **Pongono e tolgono parole dallo stack del processo**
  - Modificano lo stack pointer, che punta all'elemento sulla cima dello stack
  - In moltissimi casi lo stack cresce verso indirizzi bassi
  - Indichiamo con W la dimensione in byte della parola



- **Pone un immediato sullo stack**
  - Decrementa lo stack pointer

```
PUSH IMM          ! SP <- [SP] - W
                  ! [SP] <- IMM
```

- **Rimuove l'elemento sulla cima dello stack**
  - Incrementa lo stack pointer

```
POP Rs            ! Rs <- [[SP]]
                  ! SP <- [SP] + W
```

Operazioni di chiamata e ritorno da funzione

- **Modificano il flusso di esecuzione del programma**
  - Modificano il contenuto del program counter
  - Salvano o recuperano il valore del program counter sullo stack
  - L'utilizzo corretto di queste istruzioni richiede
    - La definizione di un meccanismo di chiamata di funzione, detto "calling convention"
    - I parametri attuali ed il valore di ritorno devono essere gestiti esplicitamente
- **Chiamata di funzione**
  - Il nome della funzione è una costante indicata da un'etichetta

```
CALL Lk          ! SP <- [SP] - W
                 ! [SP] <- PC
                 ! PC <- Lk
```

- **Ritorno da funzione**
  - Lo stack pointer deve puntare alla posizione in cui era stato salvato il program counter

```
RET              ! PC <- [[SP]]
                 ! SP <- [SP] + W
```

Operazioni speciali

- **Ogni linguaggio assembly reale dispone di alcune istruzioni speciali**
  - Tali operazioni sono molto specifiche
  - I dettagli dell'operazione svolta dipendono dall'architettura del processore
- **Operazione nulla**
  - Non esegue alcuna operazione (richiede comunque un "ciclo istruzione")

```
NOP                ! Does nothing
```

- **Chiamata di sistema**
  - Enta in modalità privilegiata (modo S)
  - Esegue una funzione di sistema operativo, specificata mediante un valore intero

```
SVC   IMM          ! Executes system function with index IMM
```

- **Ritorno da interruzione**
  - E' una istruzione simile a RET, ma deve essere usata solo nelle interrupt service routine

```
IRET               ! Returns from ISR
```

Operazioni speciali

- **Abilitazione e disabilitazione degli interrupt**
  - Non esegue alcuna operazione (richiede comunque un "ciclo istruzione")

```
DI                ! Disables interrupts
EI                ! Enables interrupts
```

- **Arresto dell'esecuzione**
  - Termina l'esecuzione e pone il processore in uno speciale stato
  - Nessuna istruzione può forzare l'uscita da tale stato
  - E' necessario un reset hardware

```
HALT              ! Stops the microprocessor
```

Modalità di indirizzamento

- **Gli operandi delle istruzioni assembly possono riferirsi ai dati in diversi modi**
- **Abbiamo già visto alcuni modi**
  - Immediati, i registri e alcuni casi di riferimenti a memoria
- **Tali modi sono detti "modalità di indirizzamento" o "addressing mode"**
- **Nel seguito useremo la seguente notazione**
  - **`W`** Dimensione della parola
  - **`IMM`** Valore immediato
  - **`ADDR`** Indirizzo assoluto
  - **`OFF`** Spiazzamento
  - **`Rn`** Registro generico
  - **`PC`** Program counter
  - **`{SRC}`** Operando sorgente, ovvero il dato che l'istruzione utilizzerà
  - **`{DST}`** Operando destinazione
  - **`{EA}`** Indirizzo effettivo dell'operando usato dall'istruzione

Modalità di indirizzamento

- **La seguente tabella riassume le principali modalità di indirizzamento**
  - Alcuni microprocessori dispongono di ulteriori modalità particolari

Modalità	Sintassi	Effetto
Immediato	`#IMM`	`{SRC} = IMM`
Registro	`Rn`	`{SRC} = [Rn]` oppure `{DST} = Rn`
Assoluto	`ADDR`	`{EA} = ADDR`
Indiretto da registro	`(Rn)`	`{EA} = [Rn]`
Indiretto da memoria	`(ADDR)`	`{EA} = [ADDR]`
Indiretto con indice e spiazzamento	`OFF(Ri)`	`{EA} = [Ri] + OFF`
Indiretto con base e indice	`(Rb,Ri)`	`{EA} = [Rb] + [Ri]`
Indiretto con base, indice e spiazzamento	`OFF(Rb,Ri)`	`{EA} = [Rb] + [Ri] + OFF`
Relativo al program counter	`OFF(PC)`	`{EA} = [PC] + OFF`
Con autoincremento	`(Ri)+`	`{EA} = [Ri], Ri <- [Ri] + W`
Con autodecremento	`-(Ri)`	`Ri <- [Ri] - W, {EA} = [Ri]`

Direttive

- **Le direttive**
  - Non sono istruzioni assembly
  - Sono indicazioni per l'assembler
- **Direttiva EQU**

```
name    EQU     value
```

  - Assegna un nome simbolico ad un valore
  - Non riserva alcuna locazione di memoria
  - Esempio

```
SIZE    EQU     100
```

Direttive

- **Direttiva ORIGIN**

```
        ORIGIN      address
```

  - Specifica l'indirizzo di inizio del blocco di dati o codice che segue
  - Non riserva alcuna locazione di memoria
  - Esempio

```
        ORIGIN      0xF000
```

- **Direttiva DATAWORD**

```
name    DATAWORD    value
```

  - Riserva una parola di memoria
  - Inizializza la locazione di memoria con il valore specificato
  - Esempio

```
SIZE    DATAWORD    128
```

Direttive

- **Direttiva RESERVE**

```
name    RESERVE    size
```

  - Riserva una zona di memoria della dimensione in byte specificata
  - Non inizializza la memoria
  - Esempio

```
ARRAY   RESERVE    10
```

- **Direttiva END**

```
        END        address
```

  - Indica all'assembler che il testo programma assembly è terminato
  - Indica qual'è l'indirizzo della prima istruzione del programma
  - Esempio

```
        END        L0
```

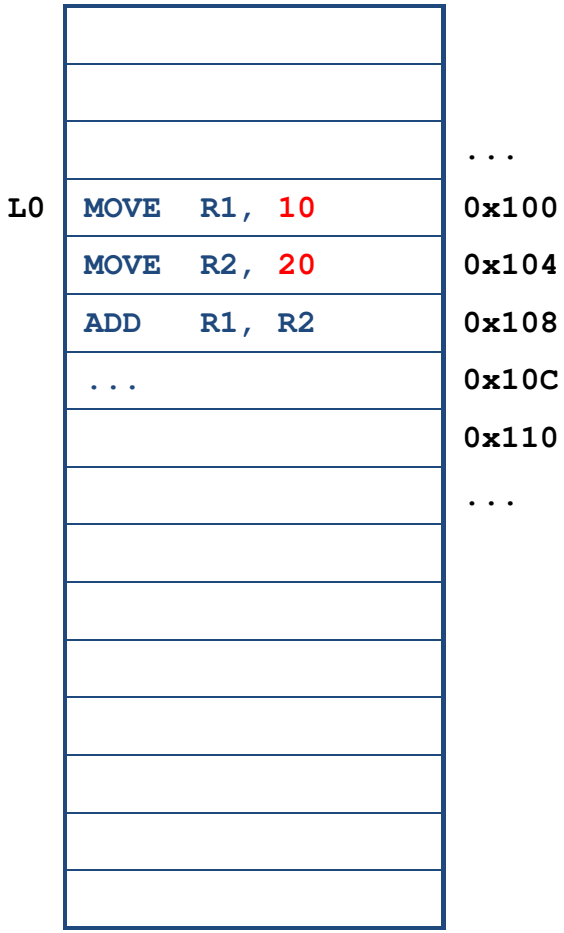
Costanti a compile time

- **Costanti a compile time**
  - Definite con la direttiva EQU
  - Equivalenti alle macro del C
- **Codice C**

```
#define A   10
#define B   20
...
C = A + B;
```

- **Codice assembly generico**

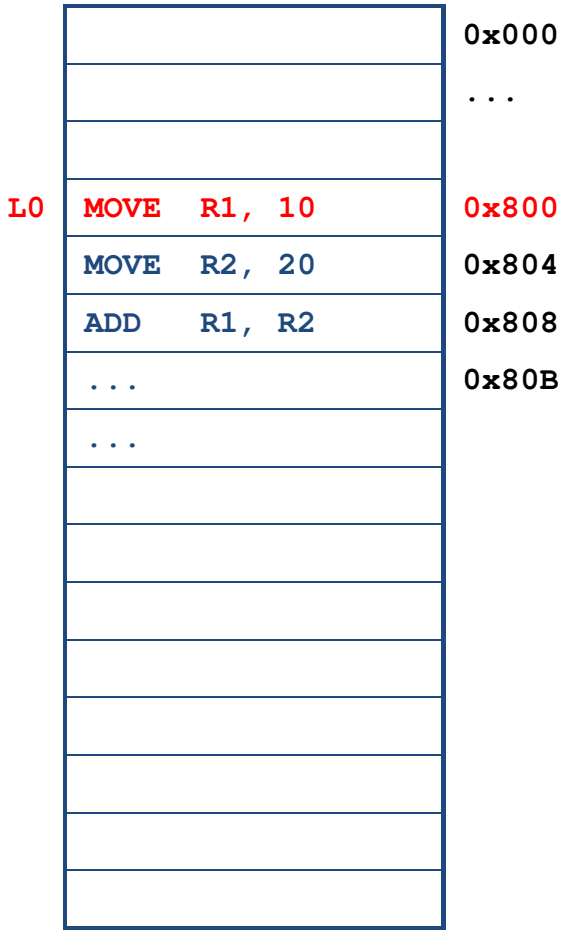
```
A       EQU         10
B       EQU         20
L0      MOVE        R1, A
        MOVE        R2, B
        ADD         R1, R2
        ...
        END         L0
```



Allocazione in memoria ad una posizione fissa

- **Posizione di dati e programma può essere fissata**
  - Si utilizza la direttiva ORIGIN
  - Assegnamento esplicito degli indirizzi
- **Codice assembly generico**

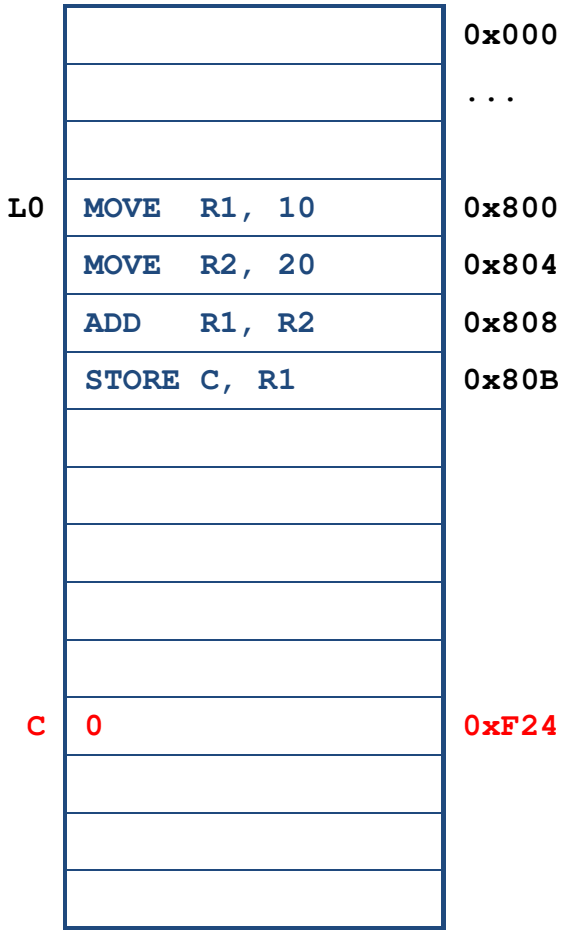
```
A       EQU         10
B       EQU         20
        ...
        ORIGIN      0x800
L0      MOVE        R1, A
        MOVE        R2, B
        ADD         R1, R2
        ...
        END         L0
```



Allocazione in memoria di dati

- **Riserva spazio in memoria per i dati globali**
  - Si utilizza la direttiva DATAWORD
  - Assegnamento esplicito di un valore iniziale
  - I dati così dichiarati costituiscono la sezione .data
  - La posizione dei dati in memoria non è fissata
- **Codice assembly generico**

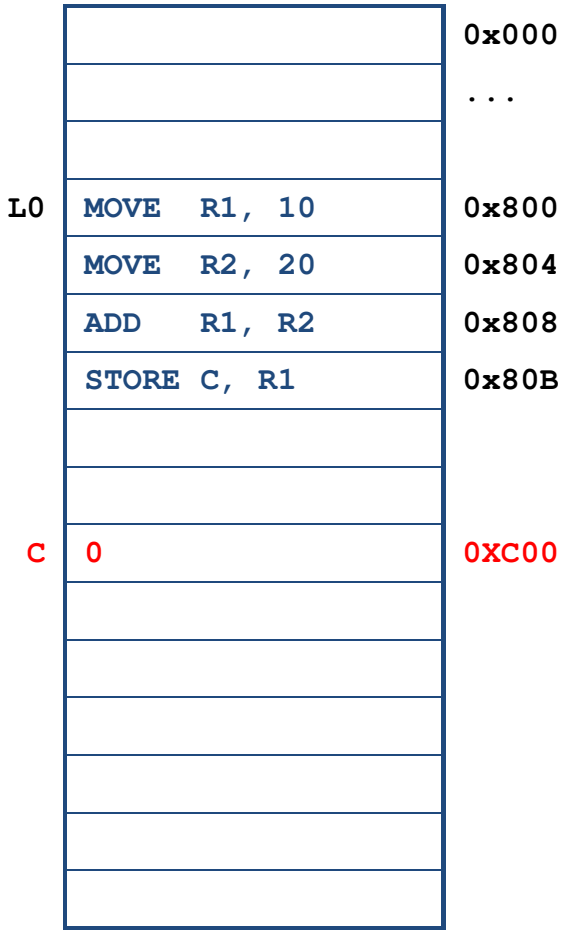
```
A       EQU         10
B       EQU         20
C       DATAWORD    0
        ORIGIN      0x800
L0      MOVE        R1, A
        MOVE        R2, B
        ADD         R1, R2
        STORE       C, R1
        END         L0
```



Allocazione in memoria di dati

- **I dati globali possono essere allocati**
  - A indirizzi specifici fissati esplicitamente
  - La direttiva ORIGIN fissa la posizione
  - LE direttiva DATAWORDpresenti nel seguito riservano spazio a partire dalla posizione indicata
- **Codice assembly generico**

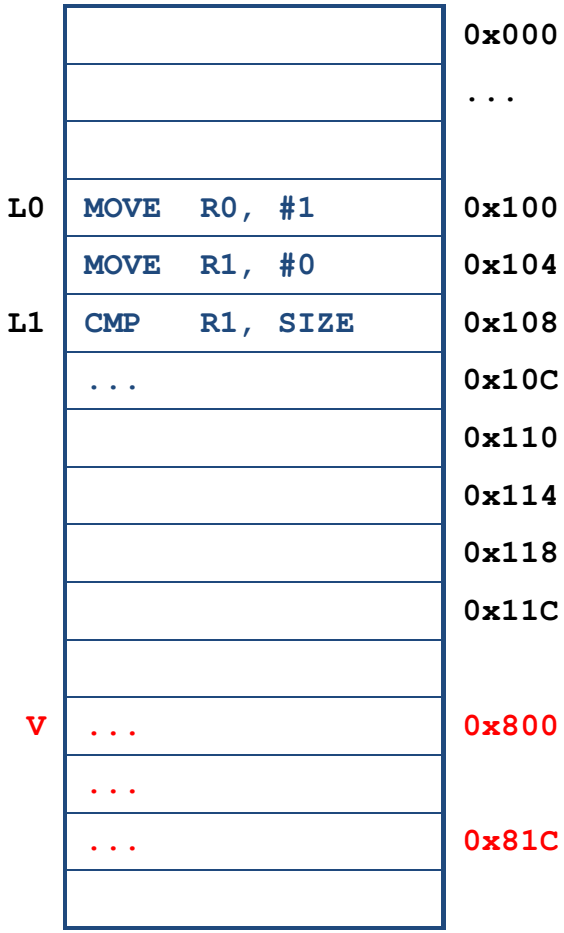
```
A       EQU         10
B       EQU         20
        ORIGIN      0xC00
C       DATAWORD    0
        ORIGIN      0x800
L0      MOVE        R1, A
        MOVE        R2, B
        ADD         R1, R2
        STORE       C, R1
        END         L0
```



Allocazione di un'area di memoria

- **Riserva una zona di memoria di dimensione arbitraria e prefissata**
  - La direttiva ORIGIN ne fissa la base
  - La direttiva RESERVE ne fissa la dimensione
  - La memoria non è inizializzata
  - Tale area di memoria appartiene alla sezione .bss
- **Codice assembly generico**

```
SIZE    EQU         8
        ORIGIN      0x800
V       RESERVE     32
        ORIGIN      0x100
L0      MOVE        R0, #1
        MOVE        R1, #0
L1      CMP         R1, SIZE
        BEQ         L2
        ...
        END         L0
```



Codice macchina

- **Il codice assembly**
  - E' detto "simbolico" in quanto leggibile facilmente dall'uomo
  - Non è eseguibile direttamente

- **Il codice macchina**
  - E' la traduzione binaria delle istruzioni assebly
  - E' eseguibile direttamente dal microprocessore
  - Contiene tutte le informazioni necessarie alla control unit per generale la sequenza di microoperazioni che svolgono la funzione richiesta

- **Il processo di traduzione da codice simbolico a codice macchina**
  - E' detto "assemblaggio"
  - E' svolto da un programma detto "assembler" o "assemblatore"
  - E' un processo di mera traduzione
    - Non richiede alcuna "intelligenza"
  - Esiste il processo inverso, detto "disassemblaggio" o "disassembly"

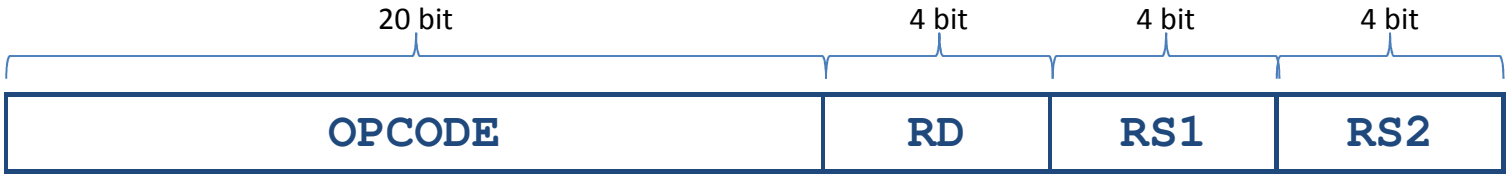
Codice macchina

- **Una istruzione macchina**
  - E' costituita da una o più parole
    - La parola ha la dimensione dei registri del microprocessore
  - Ha una struttura ben definita
    - La struttura logica è molto simile per tutti i microprocessori
    - La struttura specifica cambia da processore a processore
  - Contiene tutte le informazioni presenti nell'istruzione simbolica
- **La struttura di base è la seguente**

- **In cui**
  - OPCODE: Indica l'operazione da eseguire (ADD, SUB, ...)
  - OPERANDS: Indica gli operandi dell'operazione
  - FLAGS: Contiene informazioni ausiliarie

Codice macchina

- **Operazione su registri**
  - Assembly a tre operandi
  - 8 registri general purpose
  - 4 registri speciali
- **Ogni registro**
  - E' rappresentabile mediante un numero da 0 a 11
  - Quindi richiede 4 bit
- **Il campo opcode**
  - Può estendersi per la parte rimanente della parola



  - Può avere dimensione minore, nel qual caso si hanno bit di padding

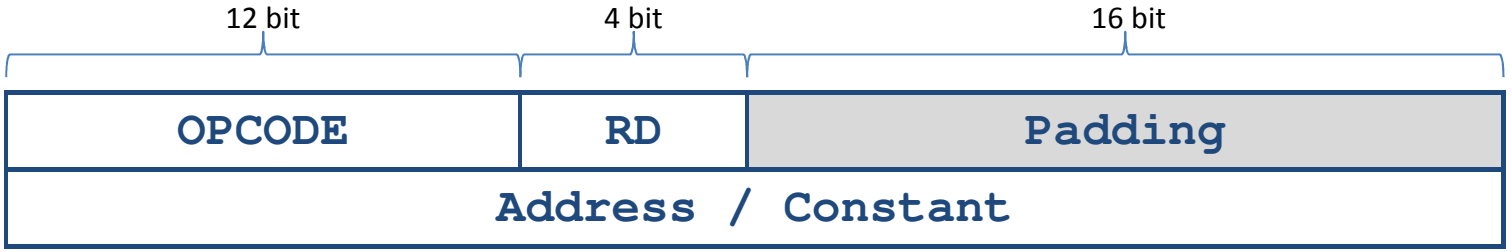
Codice macchina

- **Operazioni con dati immediati**
  - Assembly a tre operandi
  - 8 registri general purpose
  - 4 registri speciali
  - Costante numerica
- **La costante numerica può occupare al massimo lo spazio inutilizzato**

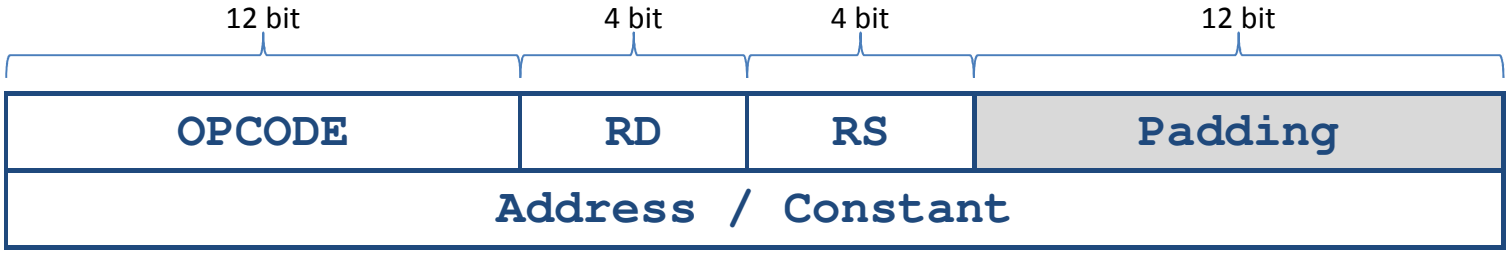
- **In questo esempio la costante numerica**
  - E' rappresentata su 12 bit
    - Quindi ha un range (0,4096) oppure (-2048,+2047)
- **Un tale formato è adatto quando le costanti sono piccole**
  - Tipicamente nel caso di spiazzamenti

Codice macchina

- **Operandi costanti di grandi dimensioni**
  - Assembly a tre operandi
  - 8 registri general purpose
  - 4 registri speciali
  - Costante numerica a 32 bit (indirizzo o valore numerico)
- **Esempio: LOAD RD, ADDR**

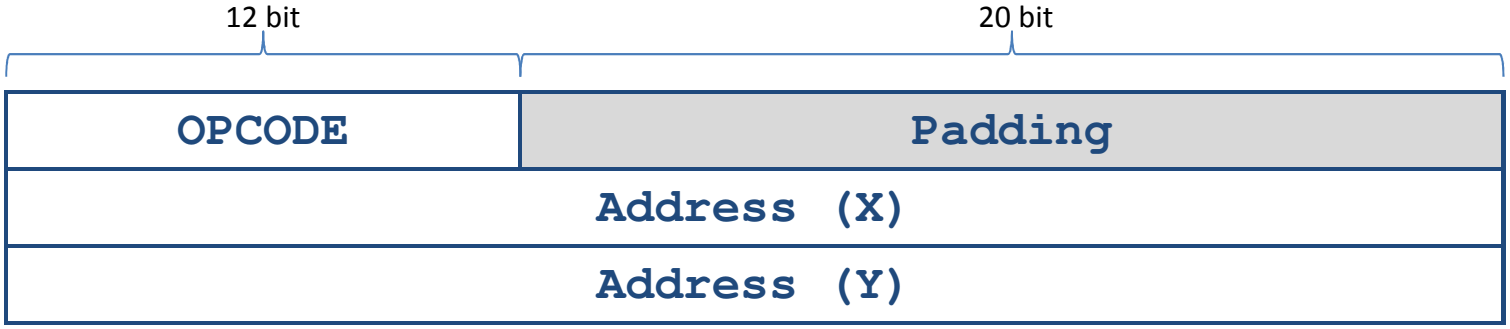


- **Esempio: LOAD RD, ADDR(RS)**



Codice macchina

- **Operazioni CISC direttamente in memoria**
  - Assembly a due operandi
  - Due indirizzi a 32 bit
- **Esempio: ADD ADDRX, ADDRY**



Esempio: Codice macchina MIPS

- **MIPS è un instruction set standard e di grande diffusione**
  - Si tratta di un linguaggio RISC
  - Pochissime istruzioni semplici
  - Linguaggio a tre operandi

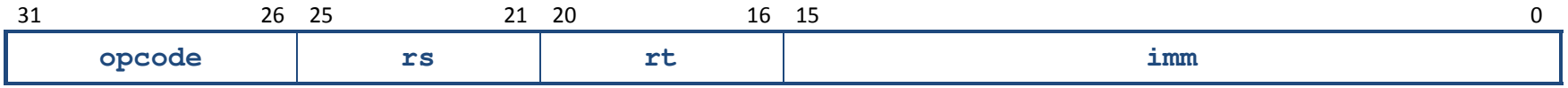
- **Le istruzioni si dividono in classi**
  - R Instructions / RF Instructions
    - Operazioni intere o floating-point fra registri
  - I Instructions / IF Istructions
    - Operazioni intere o floating-point con immediati a 16 bit
  - J Instructions
    - Istruzioni di salto con indirizzo a 26 bit

- **Ad ogni classe corrisponde uno specifico formato**
  - Tutte le istruzioni occupano una sola parola di 32 bit

Esempio: Codice macchina MIPS

- **R format: add $rd, $rt, $rs**
  - Shift specifica uno shift opzionale del risultato
  - Function specifica una variante dell'opcode

- **I format: add $rt, $rs imm**
  - Imm è un valore immediato a 16 bit



- **J format: j address**
  - Address è un indirizzo assoluto a 26 bit