

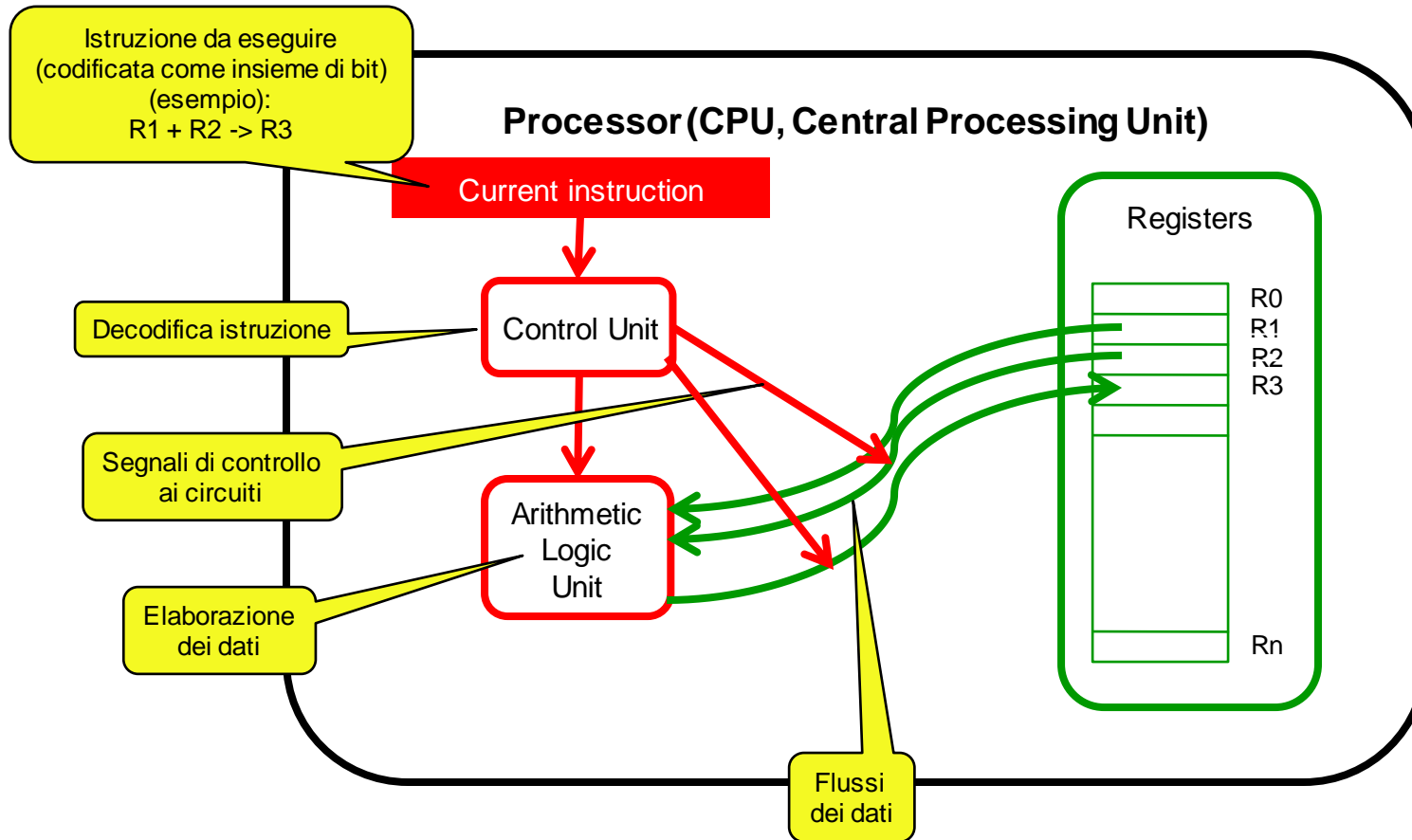
La macchina programmata

Instruction Set Architecture (2)

Istruzioni I-type
Indirizzamento della memoria
Istruzioni Load/Store
Un programma elementare

Claudia Raibulet
claudia.raibulet@unimib.it

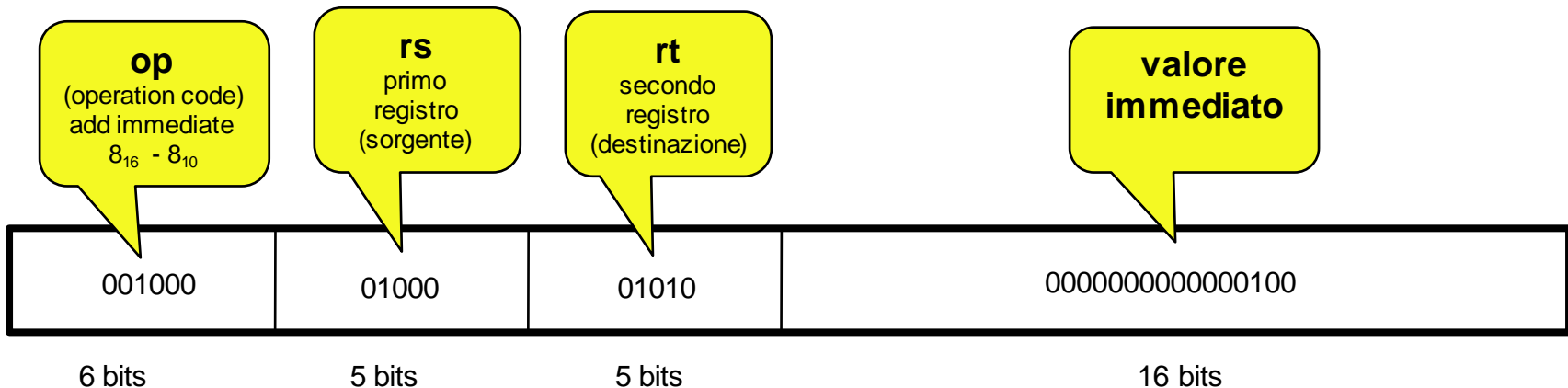
Processor: istruzione elementare su registri



Istruzione I-type (add immediate)

quello che si vuole fare (in italiano)
numeri decimali (per comodità)

“somma il valore 4 al contenuto del registro 8 e metti il risultato nel registro 10”



- se rs contiene inizialmente 64 (00000000000000000000000000001000000)
- dopo l'esecuzione rt contiene 68 (00000000000000000000000000001000100)

ASSEMBLY: addi \$10, \$8, 4

Problema: qual è il range di valori immediati che si può esprimere con 16 bit in complemento a 2?

(-32768 <= valore <= 32767). **Se serve un valore più grande, va gestito a livello programmatico.**

Indirizzamento della memoria

La dimensione della memoria indirizzabile dipende dalle dimensioni di MAR
32 bit -> 2^{32}

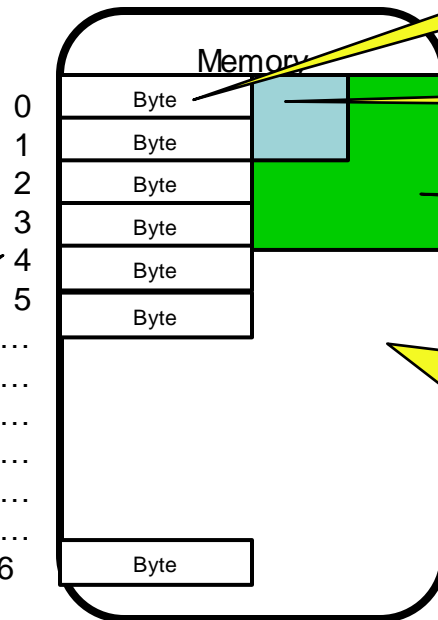
Un indirizzo è un intero positivo (Memory Address)

Word (in particolare, istruzioni) partono a multipli di 4

Commento: i 2 bit meno significativi dell'indirizzo di una istruzione sono sempre 0

Memory Address

L'unità indirizzabile è il byte (8 bit)



Half Word (16 bit)

Word (32 bit)

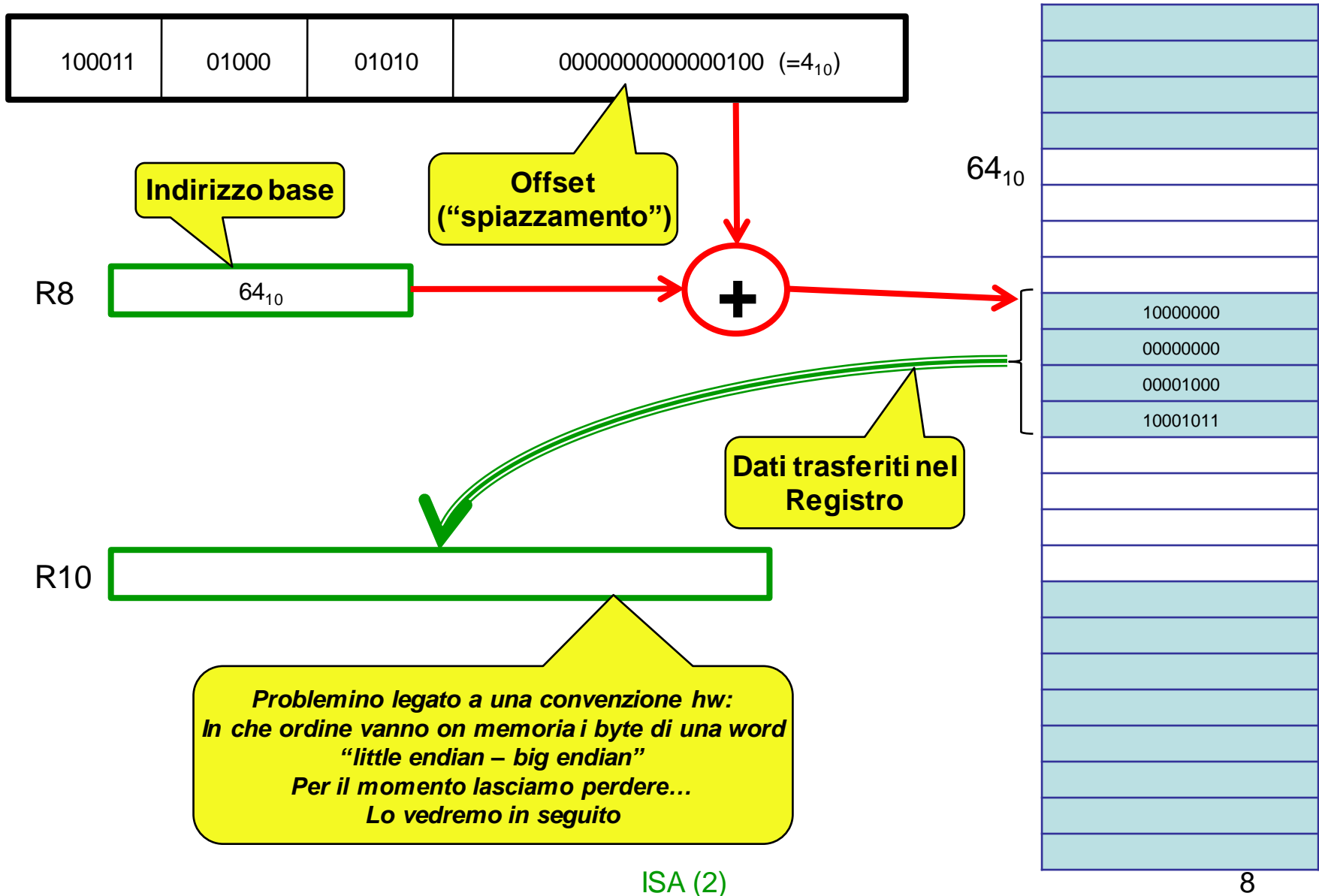
Istruzioni load/store specifiche consentono di leggere/scrivere byte, halfword, word specificando l'indirizzo del primo byte

4,294,967,296

Memory Content

- **Caricare in un registro il contenuto di una locazione di memoria**
- **Cosa *dovrebbe* specificare l'istruzione:**
 - Opcode
 - Registro destinazione
 - Indirizzo di memoria il cui contenuto viene copiato nel registro destinazione
- **Problema: quanti bit occorrono?**
 - La memoria è grande (MIPS: 2^{32} locazioni)
 - Per specificare un indirizzo occorrono 32 bit ...
 - ...che non ci stanno in una istruzione di 32 bit
- **Soluzione (MIPS):**
 - Usare il **formato I-type** (opcode, 2 registri, 1 immediato)
 - Un registro è la destinazione
 - Un registro contiene un *indirizzo base*
 - Il valore immediato è interpretato come *spiazzamento (offset)*
 - L'indirizzo cui accedere è calcolato come **(base + offset)**

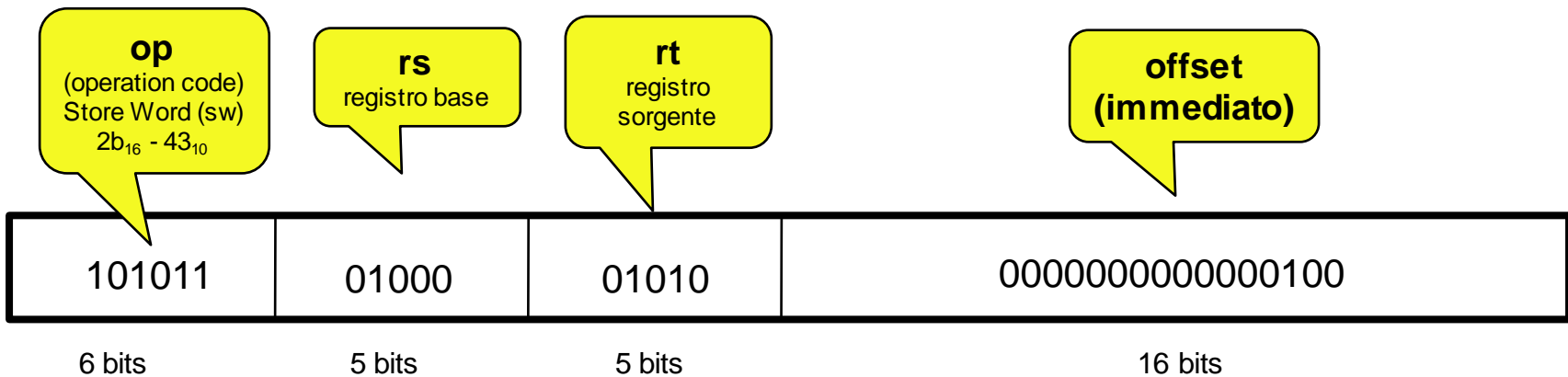
Come funziona



Istruzione store word (MIPS)

quello che si vuole fare (in italiano)
numeri decimali (per comodità)

“memorizza il contenuto del registro 10 (32 bit) all’indirizzo di memoria ottenuto come somma del contenuto del registro 8 e dell’offset immediato 4”



- confrontare con la Load Word
- ASSEMBLY: sw \$10, 4(\$8)

“carica nel registro 10 il contenuto della parola (32 bit)
che è all’indirizzo di memoria ottenuto come somma del registro 8 e dell’offset immediato 4”

lw \$10, 4(\$8)

“memorizza il contenuto del registro 10 (32 bit) all’indirizzo di memoria
ottenuto come somma del registro 8 e dell’offset immediato 4”

sw \$10, 4(\$8)

I valori relativi alle variabili a, b, c, d sono memorizzati di seguito nella memoria partendo dall'indirizzo specificato nel registro \$s0. Si chiede di scrivere la sequenza di istruzioni che aggiunge la costante 10 alle variabili a, b, c, d. I nuovi valori delle variabili vengono salvati nella memoria sempre nelle stesse locazioni di memoria.

Soluzione:

#Consideriamo, come esempio, che le variabili hanno i seguenti valori:

#a = 77, b = -5, c = 11, d = 21

#Il registro \$s0 (oppure \$16) contiene il valore 0x10008000, che rappresenta un indirizzo di memoria

lw \$s1, 0(\$s0) # leggo dalla memoria la prima variabile a in \$s1 quindi \$s1 = 77

addi \$s1, \$s1, 10 # incrementato di 10 il valore letto dalla memoria \$s1 = 87

sw \$s1, 0(\$s0) # salvo in memoria il valore incrementato

lw \$s1, 4(\$s0) # leggo dalla memoria (\$s0+4 -> 0x10008004) la seconda variabile b=-5 in \$s1=-5

ma attenzione il valore memorizzato in \$s0 resta invariato 0x10008000

addi \$s1, \$s1, 10 # incrementato di 10 il valore letto dalla memoria

sw \$s1, 0(\$s0) # salvo in memoria il valore incrementato

lw \$s1, 8(\$s0) # leggo dalla memoria (\$s0+8 -> 0x10008008) la seconda variabile b=11 in \$s1=11

addi \$s1, \$s1, 10 # incrementato di 10 il valore letto dalla memoria

sw \$s1, 8(\$s0) # salvo in memoria il valore incrementato

lw \$s1, 12(\$s0) # leggo dalla memoria (\$s0+12 -> 0x1000800c) la seconda variabile b=21 in \$s1=21

addi \$s1, \$s1, 10 # incrementato di 10 il valore letto dalla memoria

sw \$s1, 12(\$s0) # salvo in memoria il valore incrementato

Esercizio - Soluzione

I valori relativi alle variabili a, b, c, d sono memorizzati di seguito nella memoria partendo dall'indirizzo specificato nel registro \$s0. Si chiede di scrivere la sequenza di istruzioni che aggiunge la costante 10 alle variabili a, b, c, d. I nuovi valori delle variabili vengono salvati nella memoria sempre nelle stesse locazioni di memoria.

```
lw $t0, 0($s0)
addi $t0, $t0, 10
sw $t0, 0($s0)
lw $t0, 4($s0)
addi $t0, $t0, 10
sw $t0, 4($s0)
lw $t0, 8($s0)
addi $t0, $t0, 10
sw $t0, 8($s0)
lw $t0, 12($s0)
addi $t0, $t0, 10
sw $t0, 12($s0)
```

Esercizio - Osservazioni

Osservazione 1:

- Se scrivo:

```
lw $s0, 0($s0)    # sovrascrivo il valore del registro $s0 e perdo il valore
                  # dell'indirizzo di memoria dove salvare il valore incrementato;
```

- Quindi e' meglio scegliere un'altro registro e non \$s0 come registro destinazione per l'istruzione lw

Osservazione 2:

#Consideriamo come esempio che le variabili hanno i seguenti valori:

#a = 77, b = -5, c = 11, d = 21

#Il registro \$s0 oppure i\$16 contiene il valore 0x10008000, che rappresenta un indirizzo di memoria

```
lw $s1, 0($s0)    # leggo dalla memoria la prima variabile a in $s1 quindi $s1 = 77
```

```
addi $s1, $s1, 10 # incrementato di 10 il valore letto dalla memoria $s1 = 87
```

```
sw $s1, 0($s0)    # salvo in memoria il valore incrementato
```

```
addi $s0, $s0, 4  # incremento il valore dell'indirizzo memorizzato in $s0 di 4 per arrivare alla
                  # seconda variabile b; in questo modo posso usare l'offset 0 anche per la
                  # seconda lw; adesso in $s0 sara' memorizzato il valore 0x10008004
```

```
lw $s1, 0($s0)    # leggo dalla memoria ($s0+4 -> 0x10008004) la seconda variabile b in $s1
```

```
addi $s1, $s1, 10 # incrementato di 10 il valore letto dalla memoria
```

```
sw $s1, 0($s0)
```

```
addi $s0, $s0, 4  # incremento il valore dell'indirizzo memorizzato in $s0 di 4 per arrivare alla
                  # terza variabile c; adesso in $s0 sara' memorizzato il valore 0x10008008
```

A questo punto come posso tornare ad avere in \$s0 l'indirizzo di memoria relativo alla prima variabile?

Decremento \$s0 di 8 oppure uso l'offset -8.

Il costo del risotto

- **Calcolare il costo del risotto, noti i costi di riso, burro e funghi**
 - già memorizzati in variabili
- **Ci servono quattro variabili (word, in caso di inflazione)**
 - costoRiso
 - costoBurro
 - costoFunghi
 - costoRisotto
- **Algoritmo:**
 - $\text{costoRisotto} = \text{costoRiso} + \text{costoBurro} + \text{costoFunghi}$
- **Valori iniziali delle variabili:**
 - $\text{costoRiso} = 20 = 14_{16} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 0100$
 - $\text{costoBurro} = 7 = 7_{16} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111$
 - $\text{costoFunghi} = 75 = 4b_{16} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0100\ 1011$
- **Tutto questo sta su carta o in un file Word o in un file PowerPoint come questo se abbiamo un PC e Office...**
- **...ma il processore capisce solo il linguaggio macchina!**
- **Proviamo a fare a mano le cose che in pratica saranno fatte dalla “catena di programmazione” (asm + linker + loader)**

Ragionevoli dubbi

- **Dubbio 1: Chi decide dove stanno in memoria le variabili?**
 - NB: il processore ragiona SOLO in termini di indirizzi e NON CONOSCE il concetto di “nome simbolico” di una variabile!!!
 - Risposta 1a: proviamo a farlo a mano (su carta), supponendo di dover scrivere il programma in codice macchina (binario)
 - *Risposta 1b: Meccanismo Misterioso MM1: per fortuna, lo fa l'assemblatore se programiamo in linguaggio assembly*
- **Dubbio 2: Come si inizializzano in memoria i dati?**
- **Dubbio 3: Come si carica in memoria il programma?**
 - Risposte 2a e 3a: per il momento, assumiamo che esistano meccanismi misteriosi (MM2 e MM3) che lo fanno
 - *Risposta 2b e 3b: lo fa il loader (che è un programma già caricato...in maniera misteriosa)*
 - *Risposta 2c: lo fa il programma stesso se è in grado di leggere dati da una periferica e caricarli in memoria*

Allocazione delle variabili: tabella dei simboli

Le decisioni “arbitrarie” sono prese di solito in base a convenzioni usate dalla catena di programmazione ma “ignote” al processore

Decidiamo **arbitrariamente** che le variabili sono una dopo l'altra a partire da un **indirizzo base**

Definiamo l'**offset** (“spiazzamento”) di ogni variabile rispetto all'indirizzo base. NB:

- gli indirizzi sono al byte
- **nel nostro caso** tutte le variabili sono di 4 byte (word)

<i>Simbolo (nome variabile)</i>	<i>Offset₁₆</i>	<i>Init₁₆</i>
costoRiso	0 ₁₆	14 ₁₆
costoBurro	4 ₁₆	7 ₁₆
costoFunghi	8 ₁₆	4b ₁₆
costoRisotto	C ₁₆	0 ₁₆

Decidiamo **arbitrariamente** che l'indirizzo base è **1000 8000₁₆**

Siamo in grado di **calcolare l'indirizzo in memoria di ogni variabile** che poi potremo **inizializzare** (usando il meccanismo misterioso **MM3**)

Il programma!!! (frammento)

Convenzione: Registro base di Area dati

Carica indirizzo base in Registro 28
 10008000₁₆ -> Registro 28
 (Meccanismo Misterioso M4)

Carica costoRiso in Registro R7
100011 11100 00111 0000000000000000
 35 28 7 0
 23₁₆ 1c₁₆ 7₁₆ 0₁₆

Carica costoBurro in R8
 35 28 8 4
 Somma R7 e R8, metti risultato in R7
 0 7 8 7 0 32

Carica costoFunghi in R8
 35 28 8 8
 Somma R7 e R8, metti risultato in R7
 0 7 8 7 0 32

Memorizza R7 in costoRisotto
 43 28 7 12

Indirizzi

- 1000 8000₁₆
- 1000 8001₁₆
- 1000 8002₁₆
- 1000 8003₁₆
- 1000 8004₁₆
- 1000 8005₁₆
- 1000 8006₁₆
- 1000 8007₁₆
- 1000 8008₁₆
- 1000 8009₁₆
- 1000 800a₁₆
- 1000 800b₁₆
- 1000 800c₁₆
- 1000 800d₁₆
- 1000 800e₁₆
- 1000 800f₁₆

Area dati

00000000
00000000
00000000
00010100
00000000
00000000
00000000
00000000
00001111
00000000
00000000
00000000
01001011
00000000
00000000
00000000
00000000

**ISTRUZIONE
MACCHINA**

codfiche dec e hex
 dei **singoli** campi
 dell'istruzione, **non**
 dell'intera istruzione!!!

Cosa rimane da fare

- **Caricare i dati iniziali in memoria**
 - Come? Meccanismo misterioso MM2
- **Caricare il programma in memoria**
 - Dove? In questo caso elementare (senza salti!) il programma può essere caricato in qualsiasi posizione di memoria
 - ...ma vedi seguito : convenzioni ("Text Area") e istruzioni di salto
 - Come? Meccanismo misterioso MM3
- **Caricare in Program Counter l'indirizzo della prima istruzione del programma**
 - Meccanismo Misterioso MM4
- **...GO!!!**

- **Il registro base deve essere stato caricato in precedenza con un indirizzo opportuno (indirizzo base)**
- **Dove stanno i dati? Qualcuno lo deve decidere...**
- **Se i dati non occupano più di 64 kByte**
 - *il programmatore (o il compilatore o l'assemblatore) può decidere di dedicare un registro alla funzione di registro base*
 - il registro base può essere inizializzato all'inizio del programma e non viene più modificato
 - spesso il registro 28 (\$gp in Assembly MIPS) viene usato come registro base per i dati e viene inizializzato all'inizio del programma
 - è una **convenzione programmatica** sull'uso dei registri e della memoria
 - dal punto di vista hw, qualsiasi registro può essere usato come registro base e può essere modificato da programma in qualsiasi momento
- **Se (ad esempio) il registro base è inizializzato a 10008000_{16} , può essere usato come per indirizzare dati compresi fra gli indirizzi 10000000_{16} e $1000ffff$**

Continua...

- **Istruzioni di salto e di salto condizionato (salta se...)**
 - come realizzare loop e if
- **Istruzioni logiche e altre istruzioni importanti**
- ...

Esercizio

Si chiede di calcolare la somma del valore 10 che è memorizzato nel registro $\$s1$, e del valore 20 memorizzato nella locazione di memoria specificata nel registro $\$s0$ a quale si aggiunge l'offset 56 (espresso in byte). Il risultato della somma va memorizzato nella memoria 3 parole di memoria più avanti rispetto alla locazione attuale del secondo operando.

Esercizio - Soluzione

Si chiede di calcolare la somma del valore 10 che è memorizzato nel registro \$s1, e del valore 20 memorizzato nella locazione di memoria specificata nel registro \$s0 a quale si aggiunge l'offset 56 (espresso in byte). Il risultato della somma va memorizzato nella memoria 3 parole di memoria più avanti rispetto alla locazione attuale del secondo operando.

```
lw $t0, 56($s0)
```

```
add $t1, $s1, $t0
```

```
sw $t1, 68($s0)
```

```
# Offset: 56 + 3x4=68
```

Esercizio 1:

LINK: https://docs.google.com/forms/d/e/1FAIpQLSdpTAh-UeCNziNnH_RwUKIDCiK-wU-YKzApMd11DAIUmFRJPA/viewform?usp=sf_link

Esercizio 2:

https://docs.google.com/forms/d/e/1FAIpQLSe_SeX35WaJeXvLG-kyTAAI2FT7TRKzRhr7uNCu6KO6pJDJiA/viewform?usp=sf_link