Capitolo 3

Organizzazione di vCPU

In questo capito vengono affrontati tutti gli aspetti architetturali e organizzativi di vCPU. Dapprima si partirà con una visione d'insieme e successivamente si passerà ad analizzare ogni aspetto in dettaglio, come il formato delle istruzioni, l'organizzazione della memoria, i registri, i flag, la rappresentazione degli interi, la ALU e l'I/O.

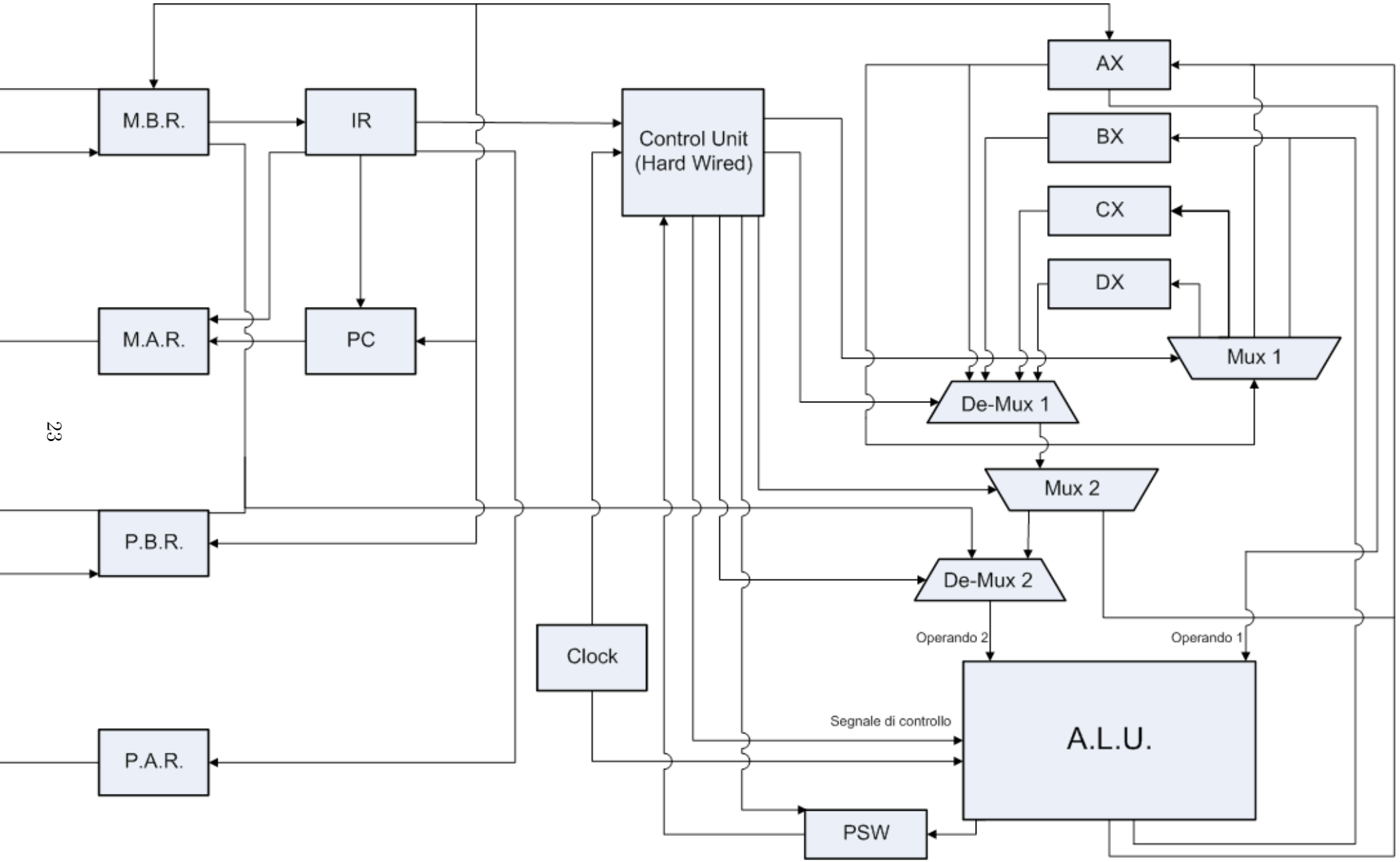
3.1 Visione d'insieme

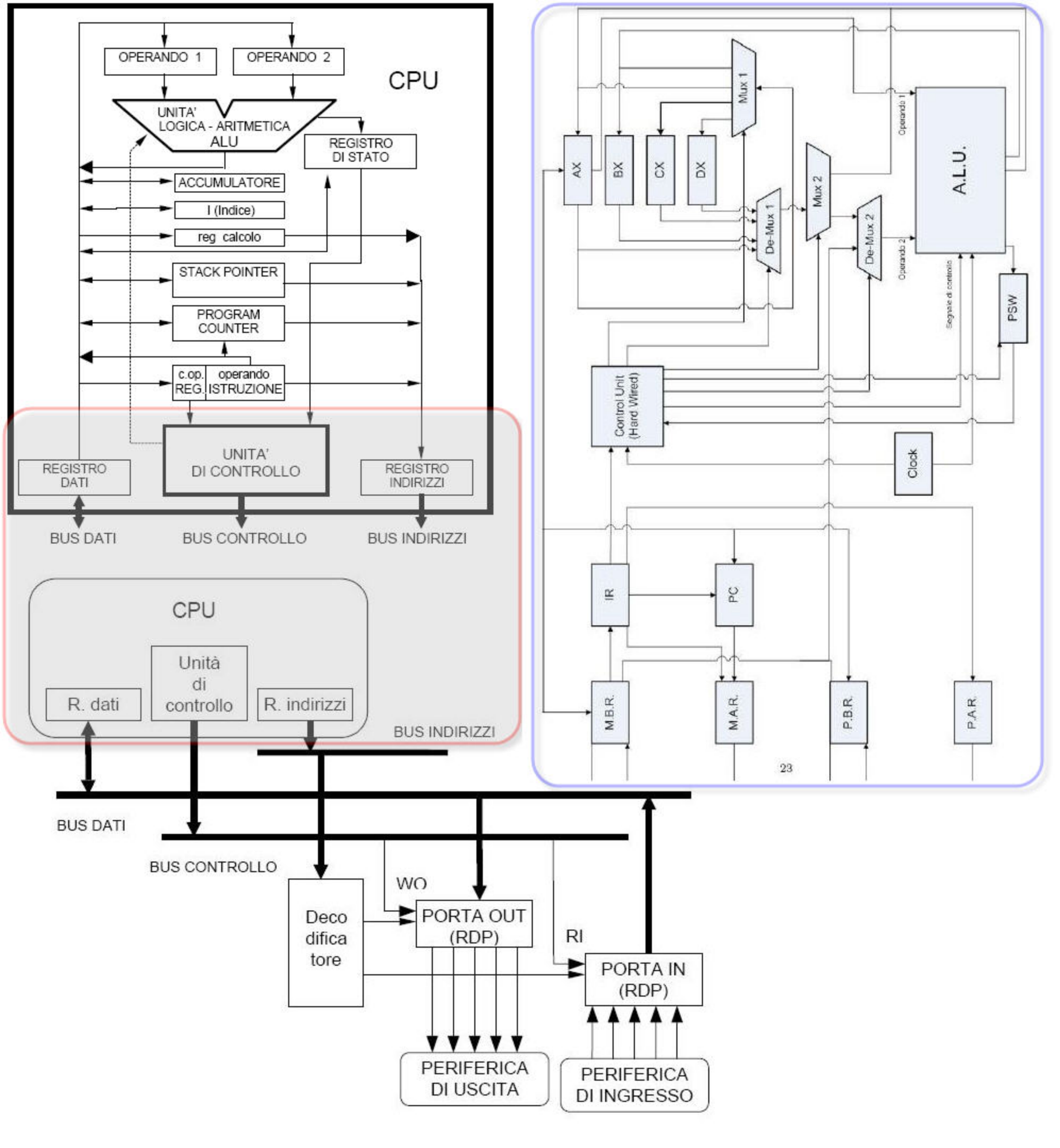
Per comprendere il funzionamento e le scelte progettuali che hanno caratterizzato i vari aspetti di vCPU, come ad esempio il set di istruzioni, è necessario analizzare la sua struttura. Nella figura 3.1 è presentato lo schema circuitale che mostra le varie componenti di vCPU e le loro relazioni.

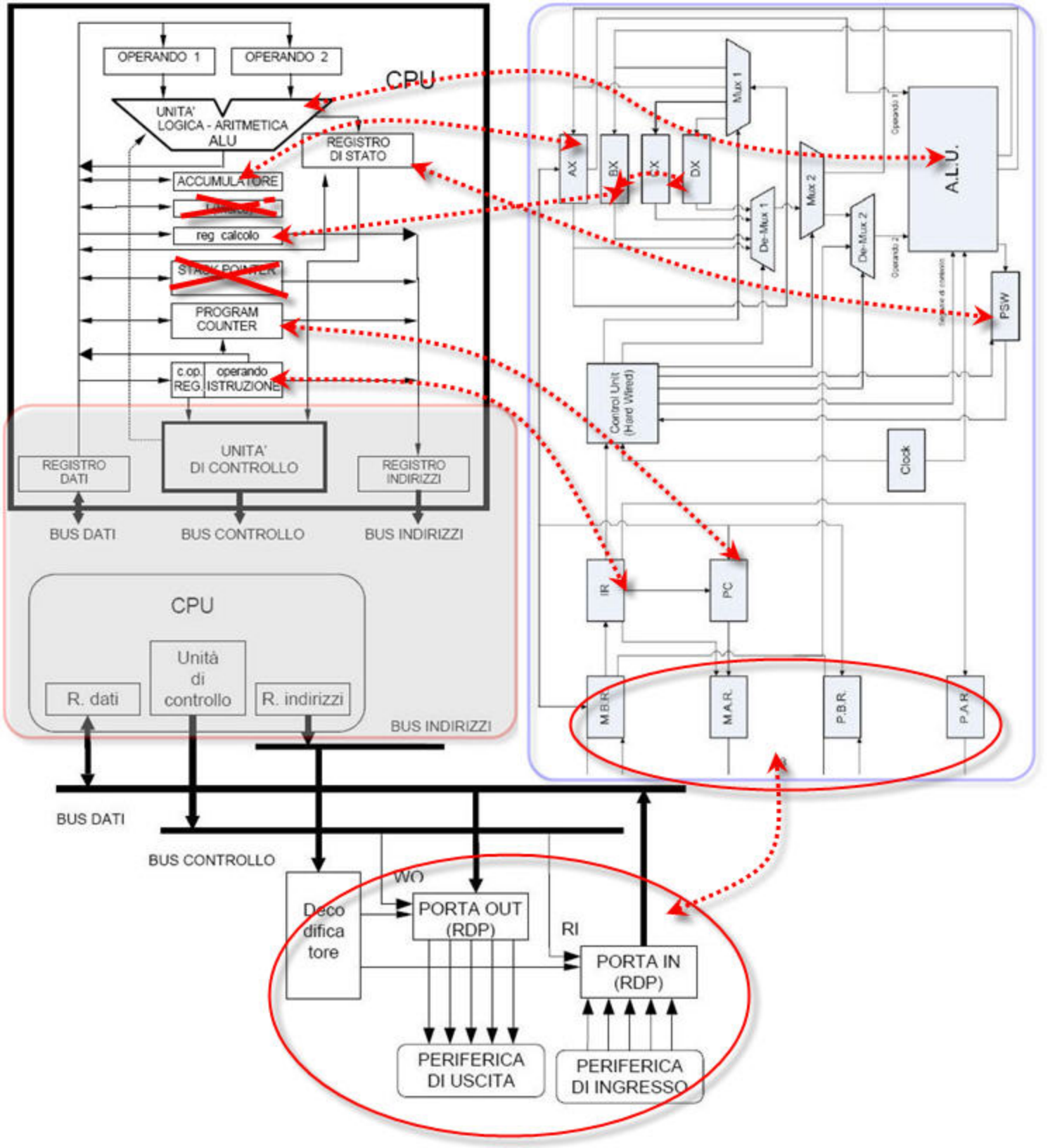
A tal proposito si noti che, essendo vCPU un'unità che non esiste nella realtà, il suo schema circuitale non è dato a priori, ma è stato definito sulla base di criteri di semplicità e significatività. In altre parole, si è scelta un'architettura non troppo complicata dal punto di vista circuitale, in modo che potesse essere compresa anche senza approfondite conoscenze di elettronica, ma sufficientemente sofisticata per permettere un reale apprendimento delle problematiche di base dell'architettura dei calcolatori.

Il punto di contatto tra la CPU e le risorse esterne, sono i registri MBR, MAR, PBR e PAR. I primi due si occupano, rispettivamente, di memorizzare il dato in transito da o verso la memoria e di contenere l'indirizzo della cella in questione; gli altri due, invece, si dedicano rispettivamente a memorizzare il dato in transito da o verso le porte ed a contenere il numero di porta al quale si sta riferendo.

l'IR (*Instruction Register*) è un registro adibito a contenere principalmente l'istruzione corrente appena prelevata dalla memoria. Il registro è collegato con la *Control Unit*, per consentire


M.B.R.
IR
Control Unit
(Hard Wired)
AX
BX
CX
DX
M.A.R.
PC
Mux 1
De-Mux 1
Mux 2
P.B.R.
De-Mux 2
Operando 2
Operando 1
Clock
Segnale di controllo
A.L.U.
P.A.R.
PSW


CPU
OPERANDO 1
OPERANDO 2
UNITA' LOGICA - ARITMETICA ALU
REGISTRO DI STATO
ACCUMULATORE
I (Indice)
reg calcolo
STACK POINTER
PROGRAM COUNTER
c.op. REG.
operando ISTRUZIONE
UNITA' DI CONTROLLO
REGISTRO DATI
REGISTRO INDIRIZZI
BUS DATI
BUS CONTROLLO
BUS INDIRIZZI
Mux 1
AX
BX
CX
DX
De-Mux 1
Mux 2
De-Mux 2
A.L.U.
Operando 1
Operando 2
Segnale di controllo
PSW
Control Unit (Hard Wired)
Clock
IR
PC
M.B.R.
M.A.R.
P.B.R.
P.A.R.
23
CPU
Unità di controllo
R. dati
R. indirizzi
BUS INDIRIZZI
BUS DATI
BUS CONTROLLO
WO
RI
Deco difica tore
PORTA OUT (RDP)
PORTA IN (RDP)
PERIFERICA DI USCITA
PERIFERICA DI INGRESSO


OPERANDO 1
OPERANDO 2
CPU
UNITA' LOGICA - ARITMETICA ALU
REGISTRO DI STATO
ACCUMULATORE
reg. calcolo
PROGRAM COUNTER
c.op. REG.
operando ISTRUZIONE
UNITA' DI CONTROLLO
REGISTRO DATI
REGISTRO INDIRIZZI
BUS DATI
BUS CONTROLLO
BUS INDIRIZZI
AX
BX
CX
DX
Mux 1
Mux 2
De-Mux 1
De-Mux 2
A.L.U.
PSW
Control Unit (Hard Wired)
Clock
IR
PC
M.B.R.
M.A.R.
P.B.R.
P.A.R.
CPU
Unità di controllo
R. dati
R. indirizzi
BUS INDIRIZZI
BUS DATI
BUS CONTROLLO
WO
RI
Deco difica tore
PORTA OUT (RDP)
PORTA IN (RDP)
PERIFERICA DI USCITA
PERIFERICA DI INGRESSO

la fase di esecuzione dell'istruzione e quindi la decodifica della stessa. Inoltre, è collegato anche con il registro PC (*Program Counter*), MAR e PAR. per poterli valorizzare con un eventuale indirizzo.

Infine, il registro PC il *Program Counter*, contiene l'indirizzo della cella contenente l'istruzione successiva da eseguire.

La CPU è fornita di ulteriori cinque registri: AX, BX, CX, DX e PSW. Il primo, AX, è un accumulatore. Ciò lascia intendere che si tratta di un registro con funzioni speciali: ad esempio, è sempre un operando e luogo di memorizzazione del risultato in ogni operazione effettuata dalla ALU. Gli altri tre registri, BX, CX, DX, sono a scopo generico: sono utilizzati per contenere informazioni. Ad esempio, possono essere utilizzati per memorizzare risultati intermedi di una lunga computazione.

Successivamente troviamo il registro PSW, adibito a contenere i flag della CPU. Come si avrà modo di approfondire nelle sezioni successive, e in particolare nella sezione 3.4, i flag hanno un ruolo fondamentale nella programmazione: sono in grado di far prendere decisioni sul flusso del programma, basandosi sull'esito di operazioni precedenti.

La *Control Unit* è il cuore della CPU: esso si occupa di decodificare l'istruzione e di eseguire l'operazione associata ad essa. Per permettergli di prendere decisioni in merito alle istruzioni precedenti, il *Control Unit* riceve in ingresso i flag memorizzati nel registro *PSW*. Da come si evince dalla figura 3.1, questa unità è in grado di veicolare il flusso delle informazioni all'interno della CPU, anche per mezzo dei multiplexer e dei de-multiplexer: ad esempio, è in grado di scegliere da quale registro prelevare il dato, per mezzo del de-multiplexer 1, e in quale registro inserire un determinato valore, per mezzo del multiplexer 1. Un'altra funzionalità importante è quella di poter controllare la ALU: tramite un segnale di controllo, è in grado di selezionare l'operazione da effettuare sugli operandi in ingresso. Permette inoltre di stabilire, tramite il de-multiplexer 2, la provenienza del secondo operando: in parcolar modo ha la possibilità di scegliere tra un registro, il PC, l'MBR, il PBR e l'IR. Il compito del multiplexer 2 è quello di permettere ai dati contenuti nei registri di essere o passati come argomento alla ALU o di essere memorizzati dentro il registro AX.

La ALU, acronimo di *Arithmetic and Logic Unit*, permette di effettuare delle operazioni sia logiche che aritmetiche. In particolare, come vedremo nella sezione 3.7, la ALU di cui dispone vCPU ha la possibilità di fare dieci operazioni differenti. Come mostrato in figura, la ALU accetta sempre come operando AX, mentre l'altro operando varia in base al segnale di controllo inviato al de-multiplexer 2. In uscita troviamo invece i flag, memorizzati nel registro PSW, e AX e BX. Come si vedrà nella sezione 3.7, il risultato tipicamente viene inserito solo in AX: BX

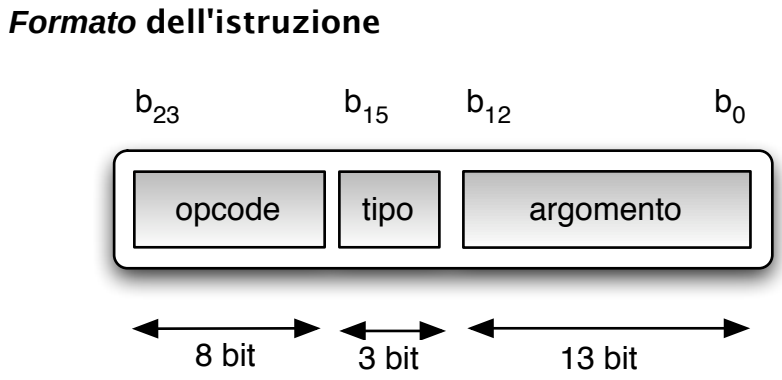


Figura 3.2: Formato dell' istruzione

è utilizzato per operazioni che richiedono maggiore spazio per memorizzare il risultato, come ad esempio l'operazione di moltiplicazione.

3.2 Formato delle istruzioni

Nei casi reali il formato delle istruzioni di una CPU è legato strettamente alla sua architettura e a vincoli tecnologici. Data la natura didattica di vCPU possiamo invece progettarlo in modo relativamente indipendente, dal momento che deve necessariamente essere compatibile con lo schema circuitale utilizzato. Ad esempio, la scelta architetturale di avere uno degli ingressi della ALU sempre costituito dal registro AX, chiaramente rende poco utile progettare un formato delle istruzioni con spazio per due argomenti, dal momento che uno dei due argomenti deve essere obbligatoriamente il registro AX.

Puntando alla semplicità, abbiamo quindi scelto un formato delle istruzioni a 24 bit. Su questi 24 bit, ne dedichiamo 8 per il *codice operativo* (detto anche *opcode*, che specifica quale operazione deve essere effettuata), e 16 bit per specificare l'argomento. Di questi, ne usiamo 3 per il *tipo* (che specifica la natura dell'argomento) e gli altri 13 per *l'argomento* (che specifica con quale operando deve essere effettuata l'operazione). (fig. 3.2). L'argomento di una istruzione può essere infatti di varia natura: potrebbe essere l'indirizzo di una cella contenente l'operando necessario all'istruzione, oppure il valore dell' operando direttamente fruibile dall'istruzione o altro ancora. Per necessità, o per loro natura se vogliamo, alcune istruzioni possono usare argomenti di diversi tipi, altre usano un argomento sempre dello stesso tipo e altre ancora non necessitano della specifica dell'argomento. Da queste differenze è possibile distinguere le istruzioni in due classi:

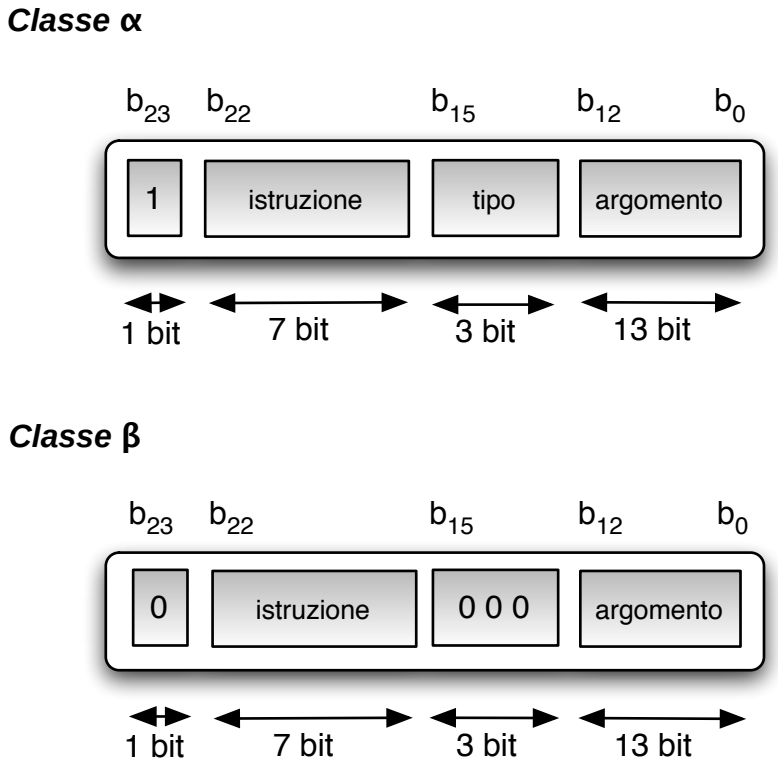


Figura 3.3: Classe α e classe β

- Argomento a tipologia variabile: istruzioni che usano argomenti di diversi tipi.
- Argomento a tipologia prefissata: istruzioni che usano un argomento sempre dello stesso tipo o che non necessitano della specifica dell'argomento.

Per rappresentare questa distinzione vengono introdotti due classi per gli 8 bit dell' opcode: la *classe* α per le istruzioni che usano argomenti di diversi tipi, e la *classe* β per quelle con argomenti a tipologia prefissata. Le classi sono distinte dal bit più significativo, b_{23}: la *classe* α e la *classe* β hanno il bit b_{23} configurato, rispettivamente, a 1 e a 0.

Come si osserva in figura 3.3, la *classe* α permette di rappresentare 2^7 istruzioni, ognuna delle quali può avere un argomento di 2^3 tipi differenti, per un totale di 2^{10} istruzioni effettive; la *classe* β permette di rappresentare solamente 2^7 istruzioni, in quanto i 3 bit di tipo, $b_{15}...b_{13}$, sono configurati sempre a 0.

3.3 Organizzazione della memoria e dimensione delle celle

Normalmente, le celle di memoria (dette anche parole) hanno una dimensione pari alla grandezza del formato delle istruzioni, molto più raramente un suo multiplo o sottomultiplo, per le

complicazioni che ciò causa nella fase di prelievo dell'istruzione. Nel nostro caso la scelta è per semplicità di 24 bit. Il numero di parole costituenti la memoria è anch'esso legato al formato dell'istruzione.

Le parole della memoria sono associate univocamente a dei valori numerici contigui, crescenti a partire da zero, chiamati **i**ndirizzi. Nel caso specifico, quindi, la memoria contiene 2^{12} parole, poiché abbiamo adottato la codifica del complemento a 2 per rappresentare gli interi e abbiamo i 13 bit dell'argomento per contenere l'indirizzo. Inoltre, un indirizzo si definisce più *basso* rispetto ad un altro se viene numericamente prima; si definisce invece più *alto* nel caso opposto.

Si vuole far notare che in teoria, grazie all' indirizzamento indiretto, sarebbe possibile fare riferimento a 2^{24} celle distinte, ma che per semplicità sono state vincolate a 2^{12} , come visto in precedenza.

3.4 I Registri

Internamente alla vCPU, sono presenti in totale sette registri:

- **AX**, l'accumulatore;
- **BX, CX, DX**, registri general purpose;
- **PSW**, il registro dei flag.
- **PC**, il program counter;
- **IR**, il registro delle istruzioni;

Ognuno di questi registri ha per semplicità di progettazione una dimensione pari a quella di una parola. AX è l'accumulatore, cioè un registro che costituisce l'operando implicito di alcune istruzioni e la destinazione implicita per la memorizzazione del risultato. I registri BX CX e DX, invece, sono utilizzatati sia per memorizzare temporaneamente informazioni, sia per essere usati come operandi nelle operazioni.

Il PSW, acronimo di *Processor Status Word*, usa ognuno dei bit in modo indipendente per rappresentare la verità o falsità di una proprietà logica di vCPU: al momento attuale ne usa solo 5 su 24, illustrati in dettaglio successivamente.

Il PC, è utilizzato per memorizzare l'indirizzo dell'istruzione successiva da eseguire. A questo registro si possono quindi assegnare degli indirizzi, ad esempio per effettuare un salto verso una cella di memoria che non sia necessariamente la seguente a quella appena eseguita.

Analogamente al registro PSW, il PC non utilizza tutti i 24 bit a disposizione, bensì i 13 meno significativi, necessari e sufficienti a rappresentare un indirizzo.

Si vuole sottolineare che il registro IR non è utilizzabile direttamente dal programmatore, in quanto usato internamente da vCPU per memorizzare l'istruzione corrente.

3.5 Rappresentazione degli interi

Uno degli aspetti principali di una CPU è la codifica adottata per la rappresentazione degli interi. Per semplicità di progettazione, vCPU è attualmente dotata di istruzioni in grado di elaborare solo interi relativi. La codifica adottata per rappresentare questi interi è quella del complemento a 2, in quanto presenta alcuni vantaggi rispetto ad altre notazioni. Questi vantaggi sono principalmente due:

- Ha una sola rappresentazione dello zero;
- Permette di effettuare le operazioni aritmetiche utilizzando la stessa circuiteria in grado di compiere operazioni aritmetiche su interi senza segno.

Un numero intero con segno (o numero relativo) X viene rappresentato dalla sequenza di cifre

$$b_n b_{n-1} ... b_1 b_0$$

dove $b_i \in \{0, 1\}$ tali che

$$-b_n 2^n + b_{n-1} 2^{n-1} + ... + b_1 2^1 + b_0 2^0 = X$$

Si rappresentano in tal modo tutti i numeri relativi che vanno da -2^n a $2^n - 1$ inclusi. A causa di questa definizione, in cui il bit significativo ha un peso negativo, la rappresentazione degli interi con segno richiede di definire a priori la grandezza dell'intervallo di rappresentazione, cioè il numero di bit dedicati alla rappresentazione del numero stesso. Ad esempio, rappresentare il numero -3 con 3 bit dà luogo a $(101)_2$, mentre rappresentandolo con 4 bit, otteniamo $(1101)_2$. D'altro canto, dato un numero rappresentato in complemento a 2 usando N bit se ne può facilmente ottenere la rappresentazione usando $N + K$ bit ($K > 0$) semplicemente aggiungendo di fronte al MSB per K volte il valore del MSB stesso. Ad esempio da $(-3)_{10} = (101)_2$ con 3 bit si passa a $(11101)_2$ con 5 bit. Si osservi, per concludere, che data una stringa di bit, ad esempio 1101, la sua interpretazione non è univoca, ma dipende dalla rappresentazione cui essa fa riferimento. Se si tratta di interi senza segno, essa equivale all'intero $(7)_{10}$, mentre se si

3.7 La ALU e le sue funzionalità

La ALU di vCPU ha la capacità di effettuare operazioni sia logiche che aritmetiche. In particolar modo, si possono eseguire operazioni di somma, sottrazione, divisione, moltiplicazione, negazione, modulo, and, or, or esclusivo (xor) e not. La ALU ha quattro ingressi (due di dati, uno di controllo e il segnale di clock) e varie uscite. ~~Come scelta progettuale, il secondo operan-~~

Capitolo 4

Il formato delle istruzioni di vCPU

In questo capitolo viene presentato e discusso in dettaglio il formato delle istruzioni e i criteri usati per assegnare gli opcode alle varie operazioni.

4.1 Argomenti e operandi

Nel Capitolo 3 avevamo informalmente introdotto i termini *argomento* e *operando*. Ne forniamo adesso una definizione:

Argomento : è il valore dei bit $b_{12}...b_0$ presenti nell'istruzione;

Operando : è il valore effettivo utilizzato dall'operazione intrapresa dall'istruzione.

Come già accennato nel Capitolo 1, le istruzioni hanno un argomento che può essere di diversa natura. Nella vCPU ci sono 5 tipi di argomento:

immediato : l'operando coincide con l'argomento.

registro : l'operando è il valore contenuto nel registro specificato come argomento.

indirizzo diretto : l'operando è il valore contenuto nella cella di memoria specificata come argomento.

indirizzo indiretto : l'operando è il valore contenuto nella cella di memoria il cui indirizzo è contenuto nella cella di memoria specificata come argomento.

registro indiretto : l'operando è il valore contenuto nella cella di memoria il cui indirizzo è contenuto nel registro specificato come argomento.

Come si vede dagli esempi, il prefisso '@' indica che il valore effettivo usato dall'istruzione non corrisponde direttamente all'argomento, ma si ottiene da esso. Esempi di questi cinque tipi di argomenti, per una ipotetica istruzione SOMMA_ACC che somma il valore dell'accumulatore a quello specificato dall'argomento sono:

- SOMMA_ACC 100
- SOMMA_ACC @100
- SOMMA_ACC @@100
- SOMMA_ACC BX
- SOMMA_ACC @BX

4.2 Criterio di assegnazione degli opcode alle istruzioni

Sono stati assegnati alle varie istruzioni i codici di operazione in modo razionale e in modo tale che sia facile ricordare quale sia il codice operativo di una data istruzione. Nelle sezioni seguenti verrà descritto il criterio di assegnazione degli opcode in relazione sia alla classe α che alla classe β.

4.2.1 Istruzioni nella classe α (tipologia variabile)

4.2.1.1 Codifica delle operazioni

Come abbiamo già accennato in precedenza, la classe α (bit $b_{23} = 1$) permette di rappresentare 2^7 istruzioni differenti. Queste istruzioni possono essere divise nelle seguenti categorie:

Prodotti : contiene le istruzioni relative a moltiplicazioni e divisioni;

Somme : contiene le istruzioni relative ad addizioni e sottrazioni;

Memorizzazione : contiene le istruzioni che permettono il flusso di dati da e verso la memoria;

Logiche ed altro : contiene le istruzioni relative ad operazioni logiche ed altro.

Dei 7 bit disponibili per codificare l'istruzione dedichiamo quindi i primi due, $b_{22}b_{21}$, per rappresentare la categoria mentre i successivi, $b_{20}...b_{16}$, codificano la specifica operazione all'interno della categoria come mostrato in figura 4.1. In realtà, dato il numero relativamente ridotto di

Capitolo 5

Le istruzioni di vCPU

In questo capitolo presentiamo e discutiamo tutte le istruzioni di vCPU. Esse sono suddivise nelle seguenti categorie, prescindendo dalla loro classe, per essere analizzate in base alla loro funzione:

Aritmetico logiche sono le istruzioni che effettuano operazioni sui dati: sono sia aritmetiche che logiche.

Memorizzazione sono le istruzioni che permettono il flusso di dati dai registri alla memoria e viceversa.

Test, salto e controllo del flusso del programma sono le istruzioni che possono deviare il flusso del programma.

I/O sono le istruzioni che permettono il flusso dei dati da e verso l'esterno.

Di seguito, per ogni categoria, è presente una tabella organizzata nel seguente modo: nella prima colonna troviamo il codice macchina dell'istruzione; nella seconda troviamo la sua sintassi; nella terza troviamo la semantica scritta in un semplice formalismo; nella quarta sono elencati i flag che possono essere modificati dall'esecuzione dell'istruzione; infine, nella quinta e ultima colonna, troviamo un esempio. In tali tabelle, ogni riga contiene tutte le possibili varianti di una istruzione relativamente al tipo di argomento. Nel presentare e discutere le istruzioni useremo inoltre da un punto di vista sintattico le seguenti convenzioni:

R : per indicare un nome di un registro (vedi sezione 3.4);

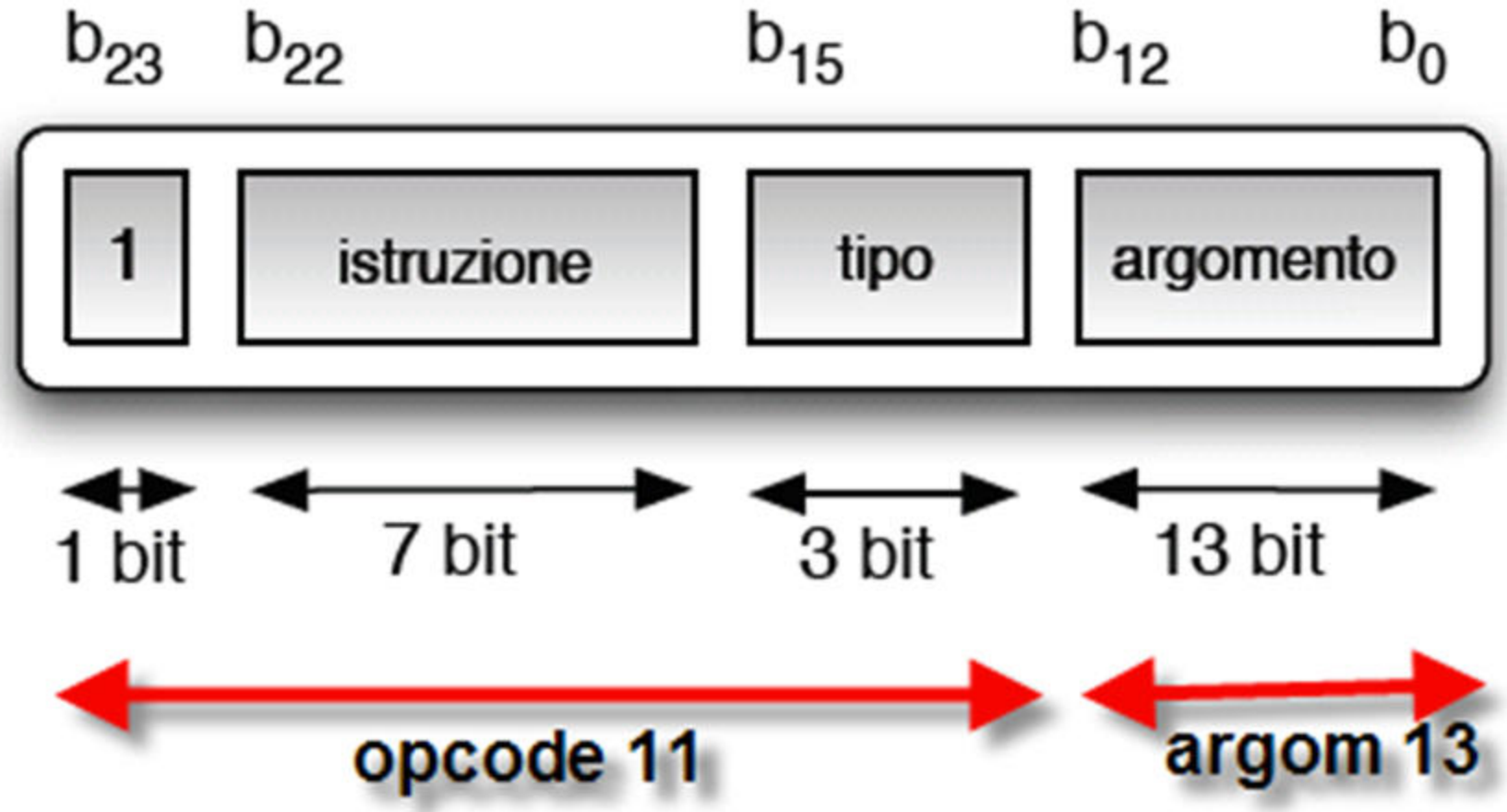
N : per indicare un intero intero relativo positivo in complemento a 2 sui 13 bit dell'argomento, appartenente quindi all'intervallo $[0, +4095]$;

Z : per indicare un intero relativo in complemento a 2 sui 13 bit dell'argomento, appartenente quindi all'intervallo $[-4096, +4095]$.

4.2.1.2 Codifica dei Registri

In molte istruzioni si ha la possibilità di specificare un registro come argomento. Per fare ciò è necessario assegnare una codifica ad ognuno di esso. La codifica adottata da vCPU è la seguente:

	b_{12}	$b_{11}...b_3$	b_2	b_1	b_0
AX	0	0...0	0	0	0
BX	0	0...0	0	0	1
CX	0	0...0	1	1	0
DX	0	0...0	1	1	1
PC	0	1...1	1	1	1


b_{23}
b_{22}
b_{15}
b_{12}
b_0
1
istruzione
tipo
argomento
1 bit
7 bit
3 bit
13 bit
opcode 11
argom 13

OPCODE E TIPO	SINTASSI	SEMANTICA	ESEMPIO
11001000000	ADD Z	$AX + Z \rightarrow AX$	ADD -10
11001000001	ADD @N	$AX + (N) \rightarrow AX$	ADD @10
11001000100	ADD R	$AX + R \rightarrow AX$	ADD BX
11000000000	SUB Z	$AX - Z \rightarrow AX$	SUB -5
11000000001	SUB @N	$AX - (N) \rightarrow AX$	SUB @10
11000000011	SUB R	$AX - R \rightarrow AX$	SUB BX
11110000000	MUL N	$AX * N \rightarrow BX : AX$	MUL 10
11110000001	MUL @N	$AX * (N) \rightarrow BX : AX$	MUL @10
11110000100	MUL R	$AX * R \rightarrow BX : AX$	MUL BX
11111000000	DIV N	$AX \ / \ N \rightarrow AX$	DIV 20
11111000001	DIV @N	$AX \ / \ (N) \rightarrow AX$	DIV @10
11111000100	DIV R	$AX \ / \ R \rightarrow AX$	DIV BX
11011000001	INC @N	$(N) + 1 \rightarrow (N)$	INC @10
11011000100	INC R	$R + 1 \rightarrow R$	INC BX
11010000001	DEC @N	$(N) - 1 \rightarrow (N)$	DEC @10
11010000100	DEC R	$R - 1 \rightarrow R$	DEC BX
01000001000	NOT	$NOT\{AX\} \rightarrow AX$	NOT
01000000000	NEG	$NOT\{AX\} + 1 \rightarrow AX$	NEG

OPCODE E TIPO	SINTASSI	SEMANTICA	ESEMPIO
10100000000	LD Z	$Z \rightarrow AX$	LD -2
10100000001	LD @N	$(N) \rightarrow AX$	LD @10
10100000100	LD R	$R \rightarrow AX$	LD BX
10111000001	ST @N	$AX \rightarrow (N)$	ST @10
10111000100	ST R	$AX \rightarrow R$	ST BX

OPCODE E TIPO	SINTASSI	SEMANTICA	ESEMPIO
00000001000	NOP	Nessuna operazione.	NOP
00000000000	HLT	Termina l'esecuzione.	HLT
00100000000	JMP N	$N \rightarrow IP$	JMP 10
00101010000	JZ N	$ZE = 1 \Longrightarrow N \rightarrow IP$	JZ 10
00101011000	JNZ N	$ZE = 0 \Longrightarrow N \rightarrow IP$	JNZ 10

01100001000	IN N	$[N] \rightarrow AX$	IN 10
01100000000	OUT N	$AX \rightarrow [N]$	OUT 10

Parte IV

Interfaccia dell'emulatore

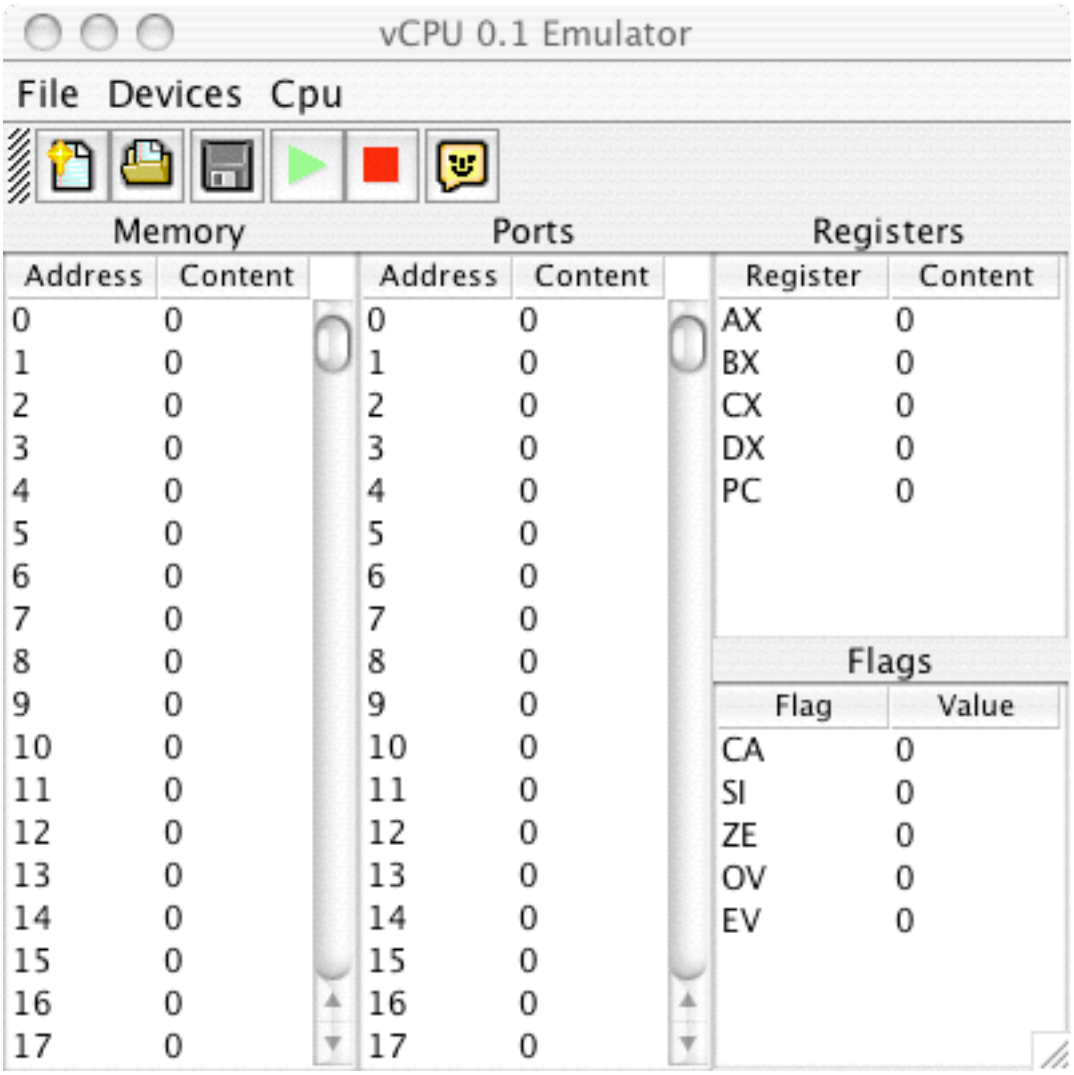


Figura 7.2: Finestra principale.

L'interfaccia grafica del prototipo dell'emulatore di vCPU è stata progettata per essere il più semplice possibile, ma al contempo anche completa per rendere l'emulatore fruibile. Tramite la realizzazione di un prototipo, è stato possibile affrontare lo studio dell'interfaccia nel modo più ampio, che verrà descritta nelle seguenti sezioni. In figura 7.2 viene mostrata l'interfaccia della finestra principale.

Capitolo 8

Finestra principale

La finestra principale si riduce alla visualizzazione dell'intero della CPU e delle risorse ad essa associate, come la memoria e le porte. L'aspetto delle varie aree della finestra sono simili e, ancor più importante, sono semplici da utilizzare: consistono in una tabella di due colonne, una per il contenuto e una per visualizzare l'etichetta associata al contenuto stesso e di un numero di righe in relazione al numero di entità presenti nella risorsa in esame. Ad esempio, sia la memoria che le porte di I/O dispongono di una tabella di due colonne e 4096 righe, ognuna delle quali rappresenta ogni singola cella.

8.1 Memoria

Come anticipato nella sezione precedente, la memoria è organizzata in una tabella di due colonne e 4096 righe, ognuna delle quali rappresenta una parola. Le parole sono ordinate in base al loro indirizzo e si presentano dalla più bassa alla più alta a partire dalla prima riga della tabella. Gli indirizzi, presenti nella prima colonna, non sono in alcun modo modificabili: rimane possibile comunque cambiare, tramite il menù del programma, il formato numerico adottato per la loro visualizzazione. Con ciò si intende che è possibile visionare gli indirizzi nel formato decimale (usato di default) ed esadecimale. Il formato binario e ottale non vengono interpellati in quanto poco utili per vari motivi: utilizzare il formato binario per visualizzare un indirizzo può essere scomodo, tanto quanto quello ottale, che inoltre è stato abbandonato dalla maggior parte dei software. In figura 8.1 è mostrata la porzione di finestra dedicata alla memoria.

Le modalità di editing della memoria possono essere molteplici: allo stato attuale è possibile inserire il codice in formato esadecimale, decimale, binario e sotto forma di stringa, nella sintassi descritta nel capitolo 5. Questo permette di arrivare a scrivere il codice in modalità, e quindi difficoltà, differenti. Il vantaggio di avere sia la possibilità di scrivere le istruzione per mezzo

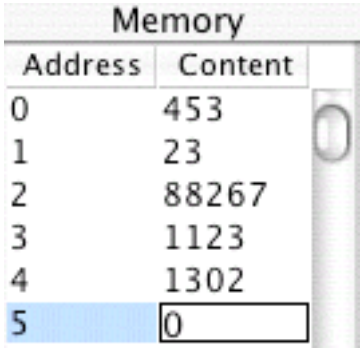


Figura 8.1: La memoria.

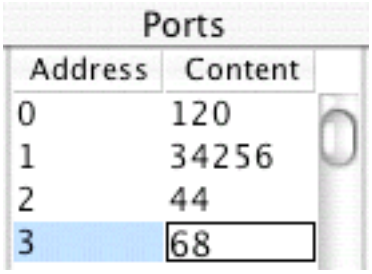


Figura 8.2: Le porte.

di una stringa e sia per mezzo del codice binario è che nel primo caso è possibile scrivere i programmi rapidamente, e nel secondo caso invece è possibile arrivare a scrivere il programma considerando la struttura e l'organizzazione dei codici operativi assegnati alle istruzioni. Il codice esadecimale gode degli stessi aspetti del codice binario, con la sola differenza che risulta più compatto e, nella maggior parte dei casi, anche più agevole.

8.2 Porte

Anche le porte adottano le stesse modalità di visualizzazione della memoria. Ciò comporta una omogeneità nell'interfaccia che risulta gradevole all'utente, in quanto non lo porta a dover apprendere diverse GUI per diverse risorse: conseguenza di ciò è una veloce comprensione dell'interfaccia grafica. Come la memoria, il valore delle porte possono essere modificate. L'unica differenza che si ha con la modalità di editing della memoria è che, ovviamente, non è possibile inserire delle stringhe, dato che non si può assegnare una istruzione come valore di porta. In figura 8.2 è mostrata la porzione di finestra dedicata alle porte.

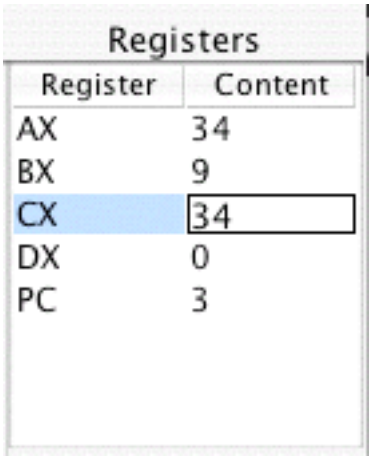


Figura 8.3: I registri.

8.3 Registri

L'area dedicata ai registri è molto più piccola rispetto a quella dedicata alla memoria e alle porte, ma con ciò non si vuole sottolineare una particolare differenza: in realtà, al posto degli indirizzi, nella prima colonna sono presenti i nomi dei registri. Analogamente alle porte, è possibile modificare il valore dei registri. I formati numerici adottati anche qui sono il binario, il decimale e l'esadecimale. In figura 8.3 è mostrata la porzione di finestra dedicata ai registri.

8.4 I Flag

Anche quest'area presenta delle somiglianze con le altre. La prima colonna, relativa alle etichette, presenta i nomi dei registri di vCPU. Gli unici valori possibili per i flag sono due: 0 e 1. Questo è dovuto al fatto che i flag vengono memorizzati in singoli bit, e pertanto le combinazioni numeriche che possono essere memorizzare sono due. Il numero ridotto di valori porta ad assumere una diversa modalità di input, effettuato per mezzo di una casella combinata con i soli valori 0 e 1 disponibili. In questo caso, avere più formati di editing è totalmente inutile, perché non si potrebbero distinguere: i valori 0 e 1 sono presenti in ogni formato numerico. In figura 8.4 è mostrata la porzione di finestra dedicata ai flag.

8.5 Plug e Unplug dei dispositivi

La possibilità di effettuare il Plug e l'Unplug dei dispositivi ha chiaramente un riscontro anche nell'interfaccia dell'emulatore. Tramite il menù *Devices* è possibile accedere ai sottomenù *Plug* e *Unplug* che rispettivamente contengono la lista dei dispositivi collegati e scollegati dal sistema.

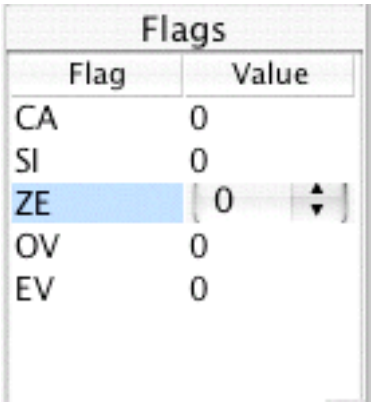


Figura 8.4: I flag.

Una volta che si decide di collegare un dispositivo al sistema, è necessario selezionare, come anticipato prima, il menù *Devices* e il sottomenù *Plug*. Qui si trovano una serie di voci di menù, ognuna delle quali rappresenta un determinato dispositivo. Selezionando una delle voci, si provoca l'attivazione del medesimo. Ciò comporta il lancio di un thread separato, che agisce da periferica: verifica eventuali cambiamenti sulle porte, scrive i risultati su delle altre, e così via. Una volta attivato un dispositivo, la voce ad esso associata si sposta nel sottomenu *Unplug* per permettere di scollegarla in qualsiasi momento. Una volta deciso di staccare un dispositivo, si deve selezionare la relativa voce nel menù *Unplugged* ed essa ritornerà nel sottomenù *Plugged*, pronta per essere nuovamente collegata.

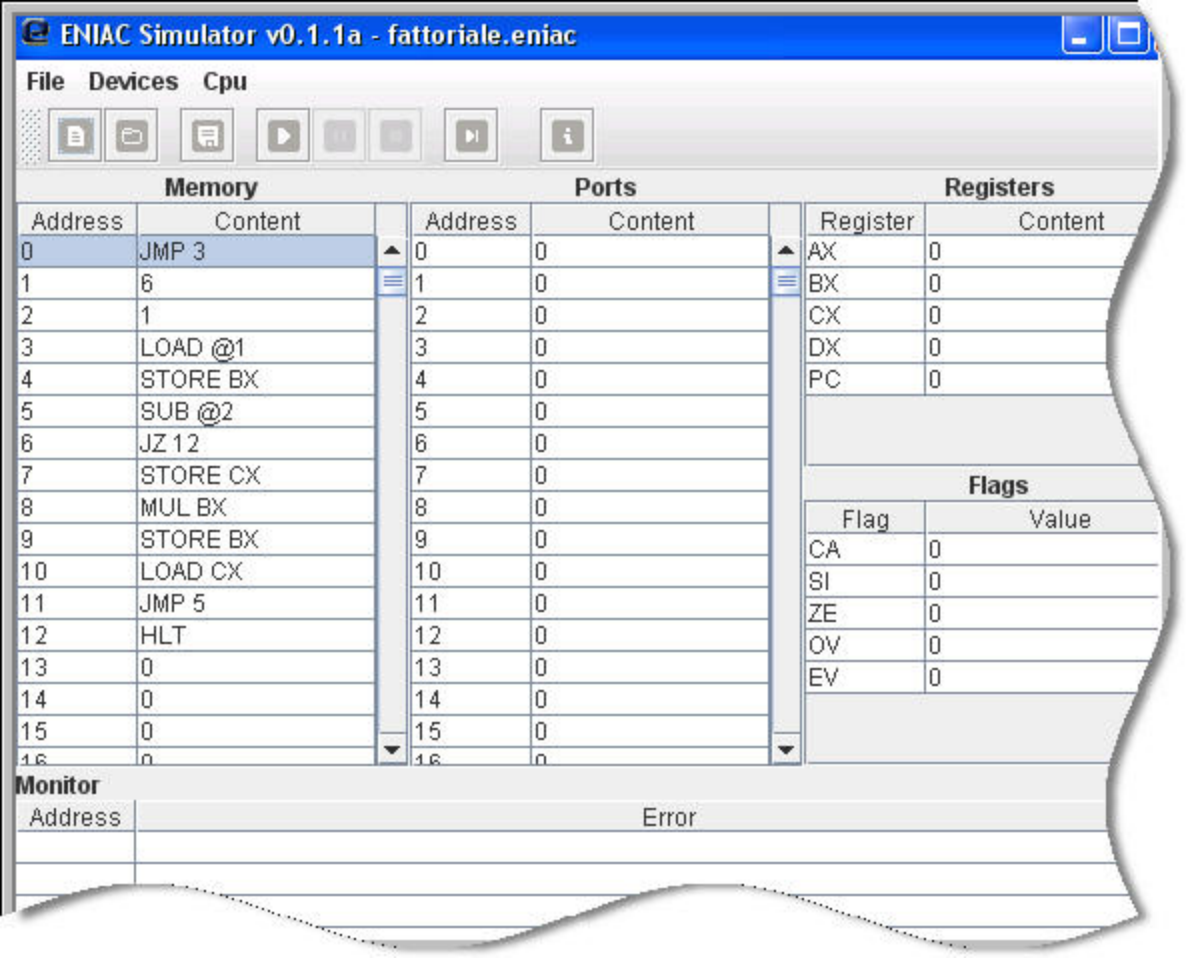
8.6 Esecuzione dei programmi

Come ogni emulatore, il software è in grado di poter eseguire dei programmi scritti in memoria. Attualmente è disponibile la modalità di esecuzione a velocità variabile, per poter permettere di osservare meglio il cambiamento dello stato della CPU.

Quando viene avviata l'esecuzione del programma da una situazione di stop, essa avviene sempre a partire dalla prima cella di memoria. Se invece si riparte dallo stato di pausa, l'esecuzione riprenderà dall'istruzione successiva all'ultima, relativamente al flusso del programma, che è stata eseguita. Per far partire, mette in pausa e fermare la CPU, sono disponibili delle voci nel menù *CPU*. Inoltre, è presente un sottomenù, *Speed*, che dispone di varie velocità, espresse in millisecondi. Scegliendone una, si vincola l'emulatore a rispettare delle pause tra l'esecuzione di una istruzione e quella successiva.

fattoriale

```
200003
6
1
a02001
b88001
c02002
2b000c
b88006
f08001
b88001
a08006
200005
10000
```


ENIAC Simulator v0.1.1a - fattoriale.eniac
File
Devices
Cpu
Memory
Address
Content
0 JMP 3
1 6
2 1
3 LOAD @1
4 STORE BX
5 SUB @2
6 JZ 12
7 STORE CX
8 MUL BX
9 STORE BX
10 LOAD CX
11 JMP 5
12 HLT
13 0
14 0
15 0
Ports
Address
Content
0 0
1 0
2 0
3 0
4 0
5 0
6 0
7 0
8 0
9 0
10 0
11 0
12 0
13 0
14 0
15 0
Registers
Register
Content
AX 0
BX 0
CX 0
DX 0
PC 0
Flags
Flag
Value
CA 0
SI 0
ZE 0
OV 0
EV 0
Monitor
Address
Error

addr		Symb	op.arg	Bin	HEX
0	vai all'inizio	JMP 3	"00100000000.0000000000011	"0010.0000.0000.0000.0000.0011	200003
1	Num	6	"00000000000000000000110	"0000.0000.0000.0000.0000.0110	000006
2	Cont	1	"00000000000000000000001	"0000.0000.0000.0000.0000.0001	000001
3	AX <-- Num	LOAD @1	"10100000001.0000000000001	"1010.0000.0010.0000.0000.0001	A02001
4	AX (ovvero Num) --> BX	STORE BX	"10111000100.0000000000001	"1011.1000.1000.0000.0000.0001	B88001
5	AX - Cont --> AX	SUB @2	"11000000001.0000000000010	"1100.0000.0010.0000.0000.0010	C02002
6	se risultato =0 vai a 12	JZ 12	"00101011000.0000000001100	"0010.1011.0000.0000.0000.1100	2B000C
7	salva il risultato in CX	STORE CX	"10111000100.0000000000110	"1011.1000.1000.0000.0000.0110	B88006
8	moltiplica AX per BX --> AX	MUL BX	"11110000100.0000000000001	"1111.0000.1000.0000.0000.0001	F08001
9	AX -->BX	STORE BX	"10111000100.0000000000001	"1011.1000.1000.0000.0000.0001	B88001
10	riprendo CX salvato in AX	LOAD CX	"10100000100.0000000000110	"1010.0000.1000.0000.0000.0110	A08006
11	ritorna al 5	JMP 5	"00100000000.0000000000101	"0010.0000.0000.0000.0000.0101	200005
12	STOP	HLT	"00000001000.0000000000000	"0000.0001.0000.0000.0000.0000	010000

dati

reg addr reg	
AX	"000
BX	"001
CX	"110
DX	"111