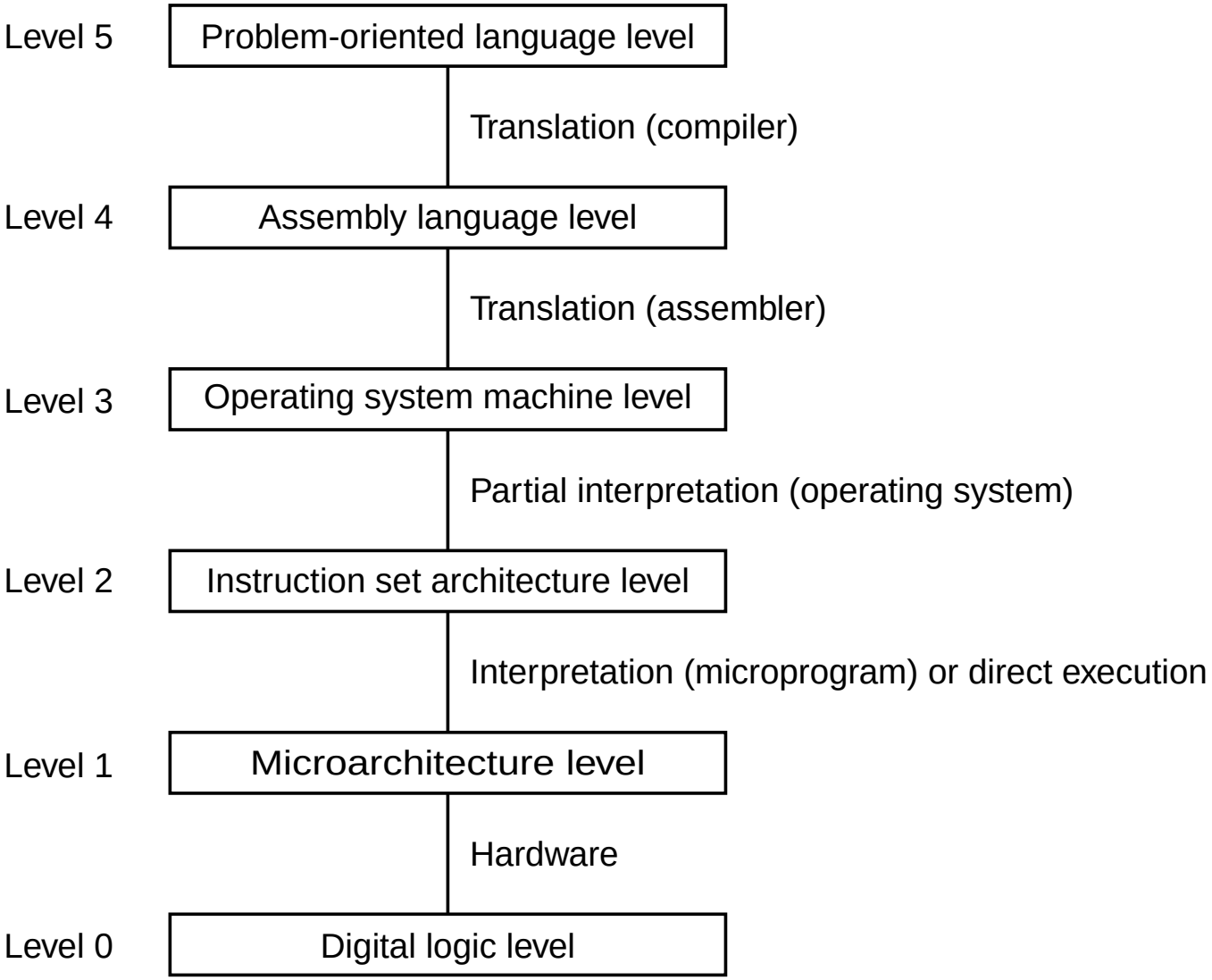
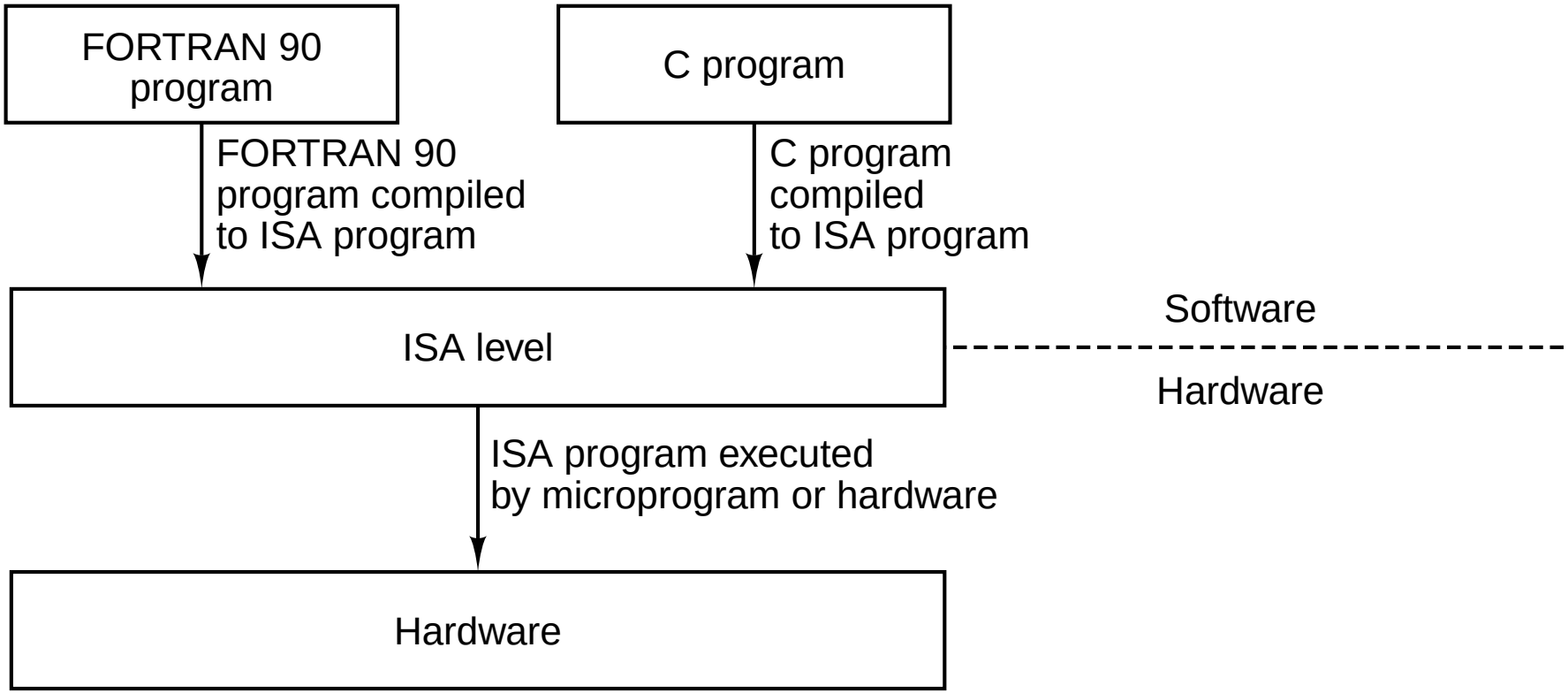
Livello 2: ISA



Instruction Set Architecture

Instruction Set Architecture (ISA): insieme delle istruzioni macchina a bordo di un processore.

ISA definisce l'interfaccia tra software e hardware.



Progettazione di un ISA

ISA sfrutterà la CPU al meglio delle possibilità:

- CISC – linguaggio macchina più simile a un linguaggio di alto livello
- RISC – poche, semplici istruzioni macchina
- VLIW (Very Long Instruction Word) – estensioni necessarie alle funzionalità più avanzate del processore (vettoriali, superscalari ecc.).

La progettazione di un ISA deve mediare i vincoli imposti al progetto hardware coi requisiti dei progetti software per la compilazione, la traduzione e l'interpretazione dei programmi utente.

Vincoli sul progetto hardware

Le istruzioni macchina devono

- sfruttare hardware veloce e a basso costo, con poche funzionalità
- essere robuste, a prova d'errore.

Vincoli sul progetto hardware

Le istruzioni macchina devono

- sfruttare hardware veloce e a basso costo, con poche funzionalità
- essere robuste, a prova d'errore.

Di conseguenza la loro espressività è limitata:

- semplici operazioni aritmetico-logiche
- poca memoria accessibile durante l'esecuzione (i registri della CPU)
- valutazione di semplici condizioni.

Solo la presenza di hardware dedicato a poche funzioni molto specifiche permette una maggiore espressività: istruzioni vettoriali (SSE, VLIW), istruzioni di salto condizionato e speculativo, ...

Il problema della retrocompatibilità

Come continuare a eseguire codice compilato per un ISA divenuto obsoleto rispetto a un hardware che nel frattempo è stato evoluto? Il problema non può essere sottovalutato dai costruttori presenti da decenni sul mercato.

IA-32 (Intel Architecture a 32 bit) della famiglia x86: Core i7 (64 bit) -> . . . -> 80386 (32 bit) -> . . . -> 80286 (16 bit) -> 8088 -> 8086, il quale a sua volta aveva parziale retrocompatibilità con 8080 (8 bit) -> 8008 -> 4004 (4 bit!)

Core i7 può (deve) emulare il processore 8088 degli anni '80! Es.: esecuzione identica di programmi MS-DOS da Windows 7.

Rinunce alla retrocompatibilità

La retrocompatibilità può essere commercialmente conveniente oppure no, a seconda del bacino di clientela target di un costruttore.

- Apple: Motorola 68000 -X> PowerPc -X> IA-32
- Sun: Motorola 68000 -X> Sparc
- Intel: Pentium-Core (IA-32) -X> Itanium (IA-64)
- Video giochi, sistemi embedded.
  Sony Play Station – PS1, PS2: MIPS, PS3: Cell, PS4: x86.

Il vincolo della retrocompatibilità

Aggiungere questo vincolo a un ISA ha il vantaggio di aumentarne la vita utile fino a vari decenni.

Per questo motivo oggi un ISA viene progettato per adattarsi allo sviluppo tecnologico. Al fine di favorire e semplificare le future estensioni, si mantiene quanta più flessibilità soprattutto in vista di aumenti

- dello spazio di memoria indirizzabile
- del numero di registri interni
- della dimensione del word
- dell'insieme di istruzioni eseguibili.

Caratteristiche di un ISA

L'ISA interfaccia il Livello 1 al Livello 3. Come interfaccia col Livello 1 ISA specifica

- tutte le funzioni del processore
- tutte le istruzioni eseguibili
- i tipi di dato comprensibili al processore
- il modello di memoria principale
- il formato di ogni istruzione
- le modalità di indirizzamento previste.

Come interfaccia col Livello 3 ISA specifica

- le modalità di funzionamento esistenti.
- la gestione dell'ingresso/uscita (I/O)

Funzioni del processore

Definiscono il comportamento del processore

- formalmente: ogni comportamento è specificato in modo normativo, attraverso una descrizione puntuale delle operazioni in cui si tralasciano quelle lasciate a specifiche customizzazioni del processore. Es.: SPARC V9, ARM
- informalmente: il comportamento è specificato in modo informativo, attraverso una descrizione tecnica che però non dischiude le operazioni puntuali. Es.: Pentium IA32.

La scelta formale o informale specifica dipende da scelte commerciali e di protezione di una tecnologia hardware e della sua complessità.

Categorie di istruzioni

Ancorchè adoperando identificatori diversi, quasi tutte le ISA specificano le seguenti categorie di istruzioni

- movimento dati con i registri e/o la memoria: `LOAD`, `STORE`, `MOVE`, ...
- operazioni aritmetiche, logiche, traslazioni e rotazioni : `ADD`, `AND`, `SHIFT_LEFT`, `ROT`, ...
- salti incondizionati e condizionati: `JUMP`, `GOTO`, `BRANCH`, `BRANCH_COND`, ...
- chiamate a procedura utente: `CALL`, `GOSUB`, `BRANCHnLINK`, `BRANCHnLINK_COND`, `RETURN`, ...
- chiamate a procedura di sistema operativo: `INVOKE`, `SYSCALL`, `SWI`.

Tipi di dati hardware

Quasi tutti i processori sono in grado di operare su alcuni dei seguenti tipi di dato.

Numerici

- naturali: unsigned 8, 16, 32, 64 bit
- interi: signed 8, 16, 32, 64 bit
- virgola mobile: float 32, 64, 128 bit
- BCD: binary code decimal format 8 bit.

Non numerici

- booleani 1 bit
- caratteri 8, 16 bit
- riferimento a indirizzi di memoria 32, 64 bit
- dati vettoriali.

Es.: tipi di dati nelle CPU Intel e ARM

Intel 32 bit (64 bit)

Type	8 Bits	16 Bits	32 Bits	64 Bits
Signed integer	×	×	×	× (64-bit)
Unsigned integer	×	×	×	× (64-bit)
Binary coded decimal integer	×			
Floating point			×	×

ARM 32 bit

Type	8 Bits	16 Bits	32 Bits	64 Bits
Signed integer	×	×	×	
Unsigned integer	×	×	×	
Binary coded decimal integer				
Floating point			×	×

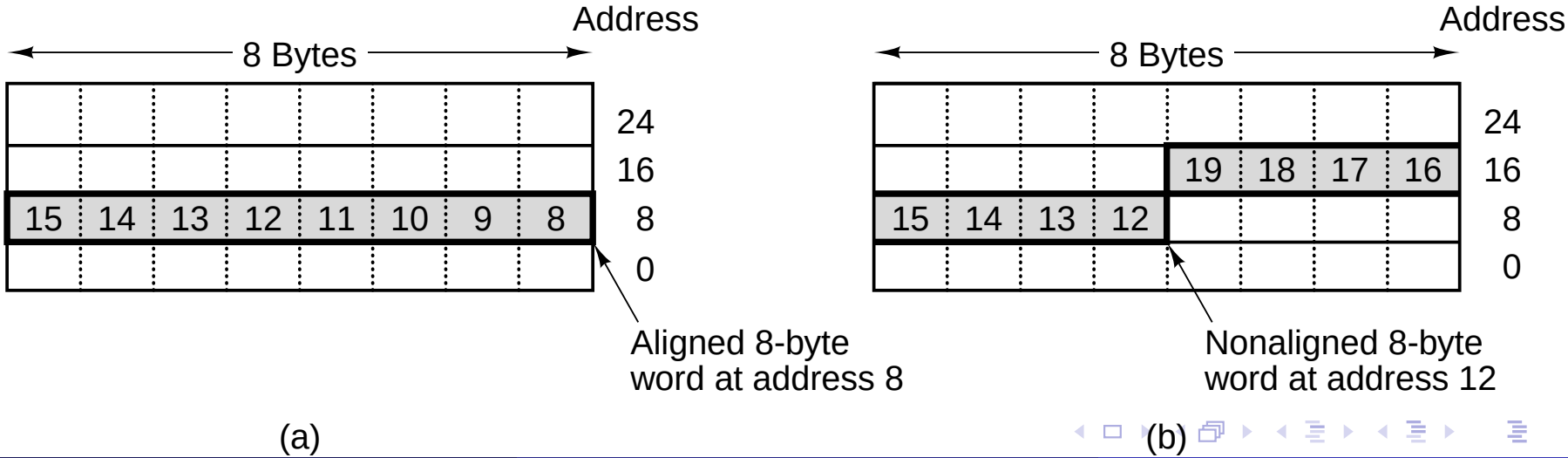
Modello di memoria principale

La memoria principale può essere logicamente unica oppure suddivisa in area istruzioni e area dati.

La locazione indirizzabile è solitamente lunga 1 byte.

Molti processori richiedono un allineamento del dato in memoria corrispondente all'estensione del tipo.

Es. (word di 8 byte): allineamento in corrispondenza di una locazione di indirizzo multiplo di 8



Modello di memoria: registri

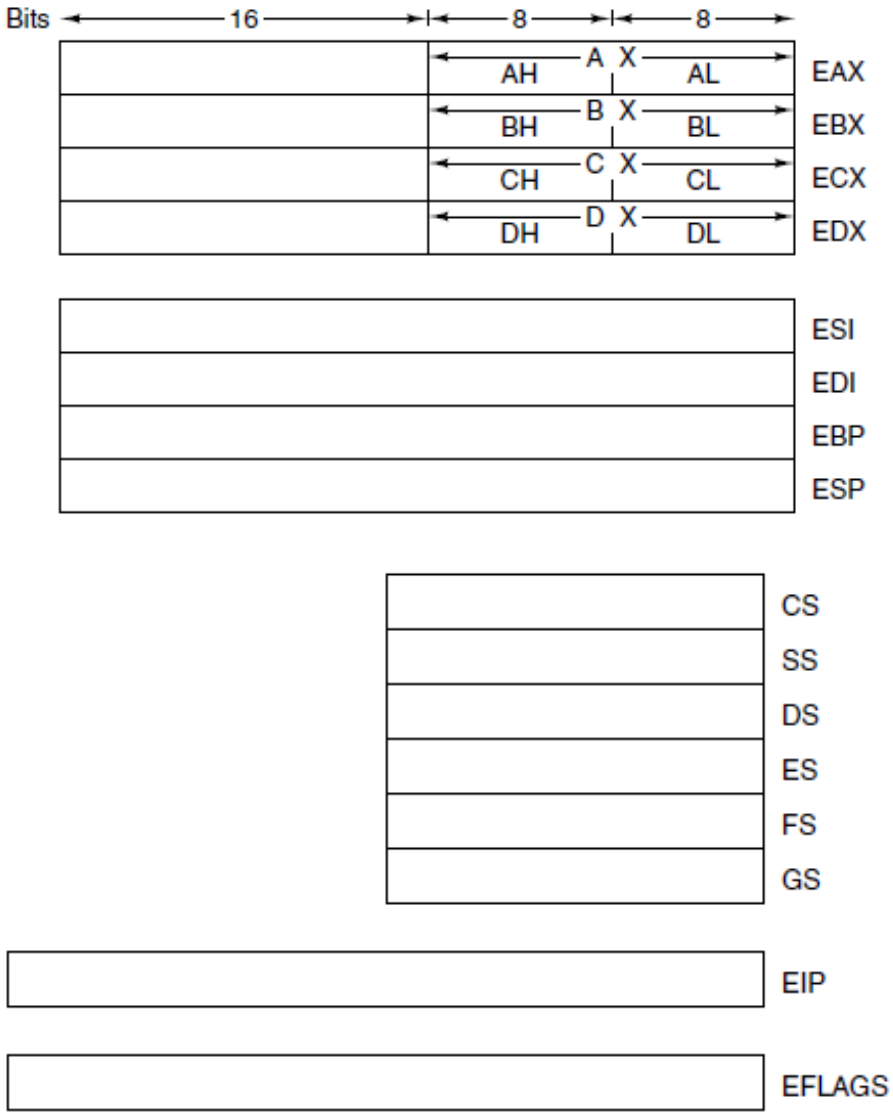
Anche i registri del processore possono essere logicamente indistinti oppure suddivisi in registri generici che memorizzano dati, e specializzati che memorizzano informazioni specifiche.

In tutti i processori troviamo almeno i registri

- PC program counter: memorizza l'indirizzo dell'istruzione successiva a quella in esecuzione
- IR instruction register: memorizza l'istruzione in esecuzione
- PSW program status word: descrive lo stato corrente del programma: bit valutati (flag) a seguito di una condizione, modalità di funzionamento, interrompibilità.

Es.: registri nella famiglia Intel x86

Tutti specializzati. Mostrati solo a titolo informativo.



Es.: registri nella famiglia ARM

ARM è un'architettura load/store: l'aritmetico-logica opera solo su dati residenti nei registri. Ogni accesso alla memoria principale richiede l'esecuzione di apposite istruzioni (`LDR`, `STR`).

Register	Alt. name	Function
R0–R3	A1–A4	Holds parameters to the procedure being called
R4–R11	V1–V8	Holds local variables for the current procedure
R12	IP	Intraprocedure call register (for 32-bit calls)
R13	SP	Stack pointer
R14	LR	Link register (return address for current function)
R15	PC	Program counter

In più ci sono 32 registri dedicati all'aritmetica in virgola mobile.

Formato delle istruzioni

Le istruzioni macchina conservano la tradizionale organizzazione data dalla sintassi (synopsis)

```
comando argomento1, argomento2, ...
```

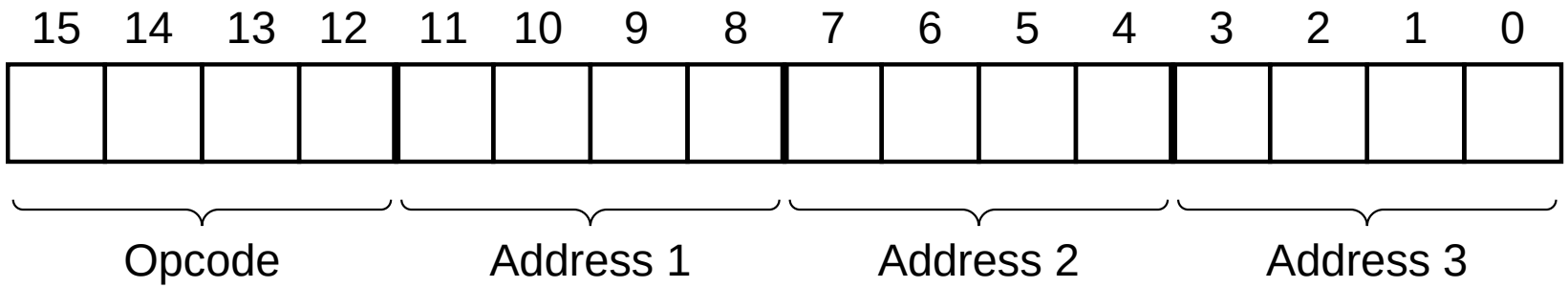
in cui il comando ordina alla CPU il tipo di elaborazione, e uno o più argomenti permettono di accedere ai dati da elaborare.

Il comando trova spazio nel campo dell'istruzione in posizione fissa, lunghezza variabile dedicato al codice operativo. Decodificato il comando dal codice operativo la CPU conosce numero e posizione degli altri campi e i tipi di dato contenuti, e può adoperarne le informazioni per eseguire l'istruzione.

Es.: istruzione a 16 bit per la somma

Poichè l'istruzione macchina è poco comprensibile, adoperiamo una sua versione in stile assembly per la vicinanza di questo linguaggio a quello macchina.

Es.: depositare in `Ra` la somma del contenuto di `Rb` + il contenuto di `Rc`: `ADD Ra, Rb, Rc`

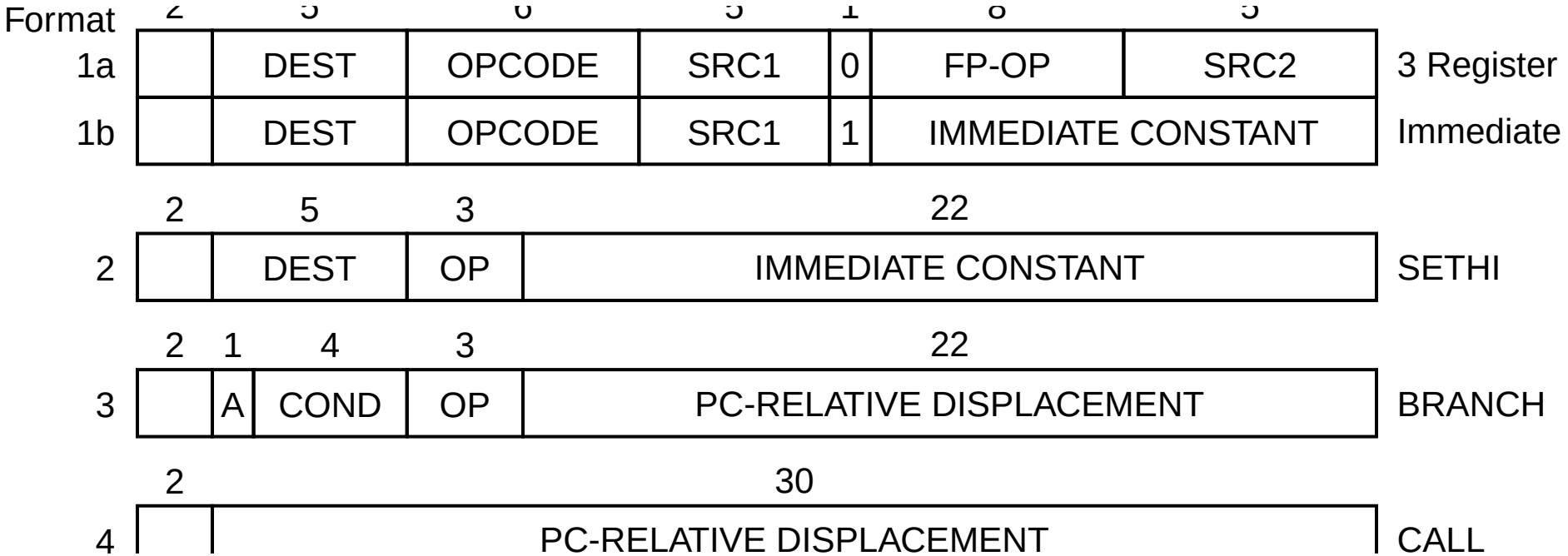


Decodificato il comando `ADD` (bit 15-12), i tre registri indirizzati dai successivi 3 campi di 4 bit (`Ra,Rb,Rc`) sono usati dalla CPU rispettivamente per depositare il risultato e caricare due valori nella ALU.

Esempio: l'ISA Sun SPARC

ISA per SPARC prevedeva a partire dal bit 24 un campo codice operativo a lunghezza variabile.

I codici operativi più brevi sono solitamente usati nelle istruzioni di salto relativo, al fine di massimizzarne l'estensione.



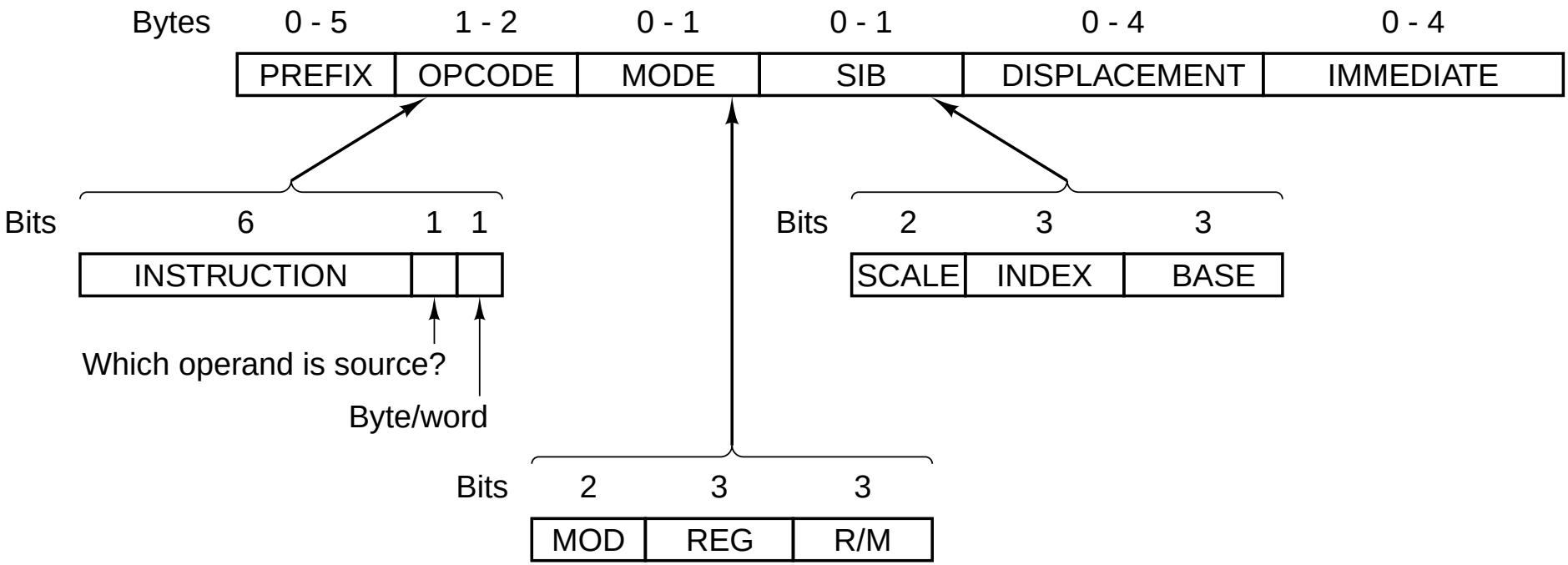
Esempio: l'ISA ARM

Elegante combinazione di codici operativi (bit 27>)

31 2827									1615		87					0	**Instruction type**
Cond	0 0	I	Opcode				S	Rn	Rd	Operand2							Data processing / PSR Transfer
Cond	0 0 0 0 0 0					A	S	Rd	Rn	RS	1 0 0 1				Rm		Multiply
Cond	0 0 0 0 1			U	A	S		RdHi	RdLo	RS	1 0 0 1				Rm		Long Multiply
Cond	0 0 0 1 0			B	0 0			Rn	Rd	0 0 0 0	1 0 0 1				Rm		Swap
Cond	0 1	I	P	U	B	W	L	Rn	Rd	Offset							Load/Store Byte/Word
Cond	1 0 0		P	U	S	W	L	Rn	Register List								Load/Store Multiple
Cond	0 0 0		P	U	1	W	L	Rn	Rd	Offset1	1	S	H	1	Offset2		Halfword transfer: Immediate offset
Cond	0 0 0		P	U	0	W	L	Rn	Rd	0 0 0 0	1	S	H	1	Rm		Halfword transfer: Register offset
Cond	1 0 1		L	Offset													Branch
Cond	0 0 0 1			0 0 1 0				1 1 1 1	1 1 1 1	1 1 1 1	0 0 0 1				Rn		Branch Exchange
Cond	1 1 0		P	U	N	W	L	Rn	CRd	CPNum	Offset						Coprocessor data transfer
Cond	1 1 1 0			Op1				CRn	CRd	CPNum	Op2	0	CRm				Coprocessor data operation
Cond	1 1 1 0			Op1			L	CRn	Rd	CPNum	Op2	1	CRm				Coprocessor register transfer
Cond	1 1 1 1			SWI Number													Software interrupt

Esempio: l'ISA x86

Solo a titolo informativo.



Argomenti di un'istruzione macchina

Al variare del modello di memoria, una stessa istruzione può essere realizzata con un numero diverso di argomenti.

Per esempio, la somma in

- ARM: tre argomenti `ADD R0, R1, R2`
- IA-32: due argomenti `ADD Ra, Rb`
- architetture (es.: processori di segnale) con accumulatore (registro dedicato all'accumulo della somma): un argomento `ADD Rx`
- Mic-1 (la memoria principale contiene anche i risultati temporanei): nessun argomento `ADD` .

Modalità d'indirizzamento

Dov'è il dato specificato in un campo argomento?
Dipende dall'indirizzamento

- immediato se il dato è nel campo: `#5`
- diretto se il dato è nella memoria indirizzata dal campo: `[0x1000]`
- di registro se il dato è nel registro specificato nel campo: `r1`
- indiretto tramite registro se il dato è nella memoria indirizzata dal registro specificato nel campo (offset): `[r1]`
- indicizzato se offset + displacement: `[r1,#100]`
- base indicizzata se anche il displacement è contenuto in un registro: `[r1,r2]`.

Esempi di indirizzamento

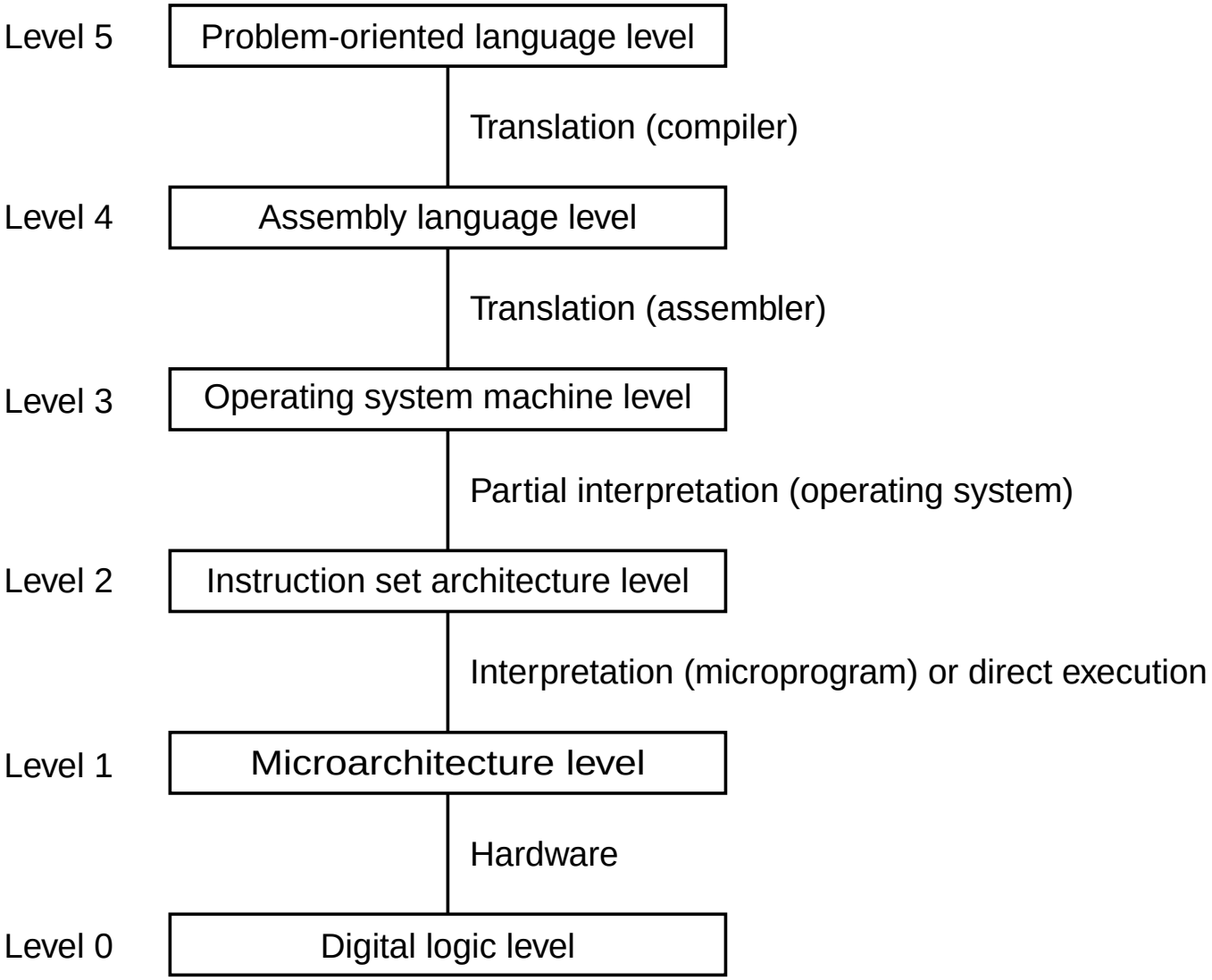
Esempi di indirizzamento negli ISA i7, ARM, Atmel

Addressing mode	Core i7	OMAP4430 ARM	ATmega168 AVR
Immediate	×	×	×
Direct	×		×
Register	×	×	×
Register indirect	×	×	×
Indexed	×	×	
Based-indexed		×	

ARM è load/store, di conseguenza non prevede l'indirizzamento diretto.

Atmel è come già visto un processore particolarmente semplice.

Livello 3: Sistema operativo



Modalità di funzionamento

Quasi ogni processore può passare a una diversa modalità di funzionamento a seguito dell'esecuzione di un'istruzione di chiamata di sistema.

Ciò è indispensabile per proteggere il sistema da richieste improprie da parte dei programmi utente.

In tal caso, il funzionamento del processore possiede almeno due modalità:

- utente, con privilegi limitati (alcune istruzioni non sono eseguibili, aree di memoria inaccessibili)
- supervisore, riservata a programmi (es.: sistema operativo) che possono eseguire qualsiasi istruzione su qualunque area di memoria.

Il passaggio di modalità

Il passaggio a una modalità più protetta (livello inferiore) può avvenire in quattro modi:

- al livello utente, a seguito di un'esplicita chiamata a subroutine di sistema.
  Es.: accesso alla memoria, stampa, eccetera
- al livello del sistema operativo, a seguito di un'esplicita chiamata di sistema.
  Es.: spostamento di una pagina di memoria
- a seguito di un evento esterno alla CPU.
  Es.: richiesta di una periferica, calo di energia
- a seguito di un evento interno alla CPU.
  Es.: divisione per zero.

Il ritorno consegue al rientro dalla routine di sistema.

Es.: funzionamento del Core i7

Modalità di funzionamento supplementari per garantire la retrocompatibilità fino all'esatta emulazione del processore 8088.

Modalità

- isa x86-64 protetta
  - livello 3: programmi utente
  - livello 2: subroutine certificate per l'utente (librerie)
  - livello 1: sistema operativo
  - livello 0: subroutine attivabili dalla sola CPU
- isa 8088 virtuale: istruzioni emulate dalla CPU (deadlock risolvibili dalla CPU)
- isa 8088 reale: istruzioni eseguite dalla CPU (deadlock della CPU reali!).

Es.: funzionamento di ARM

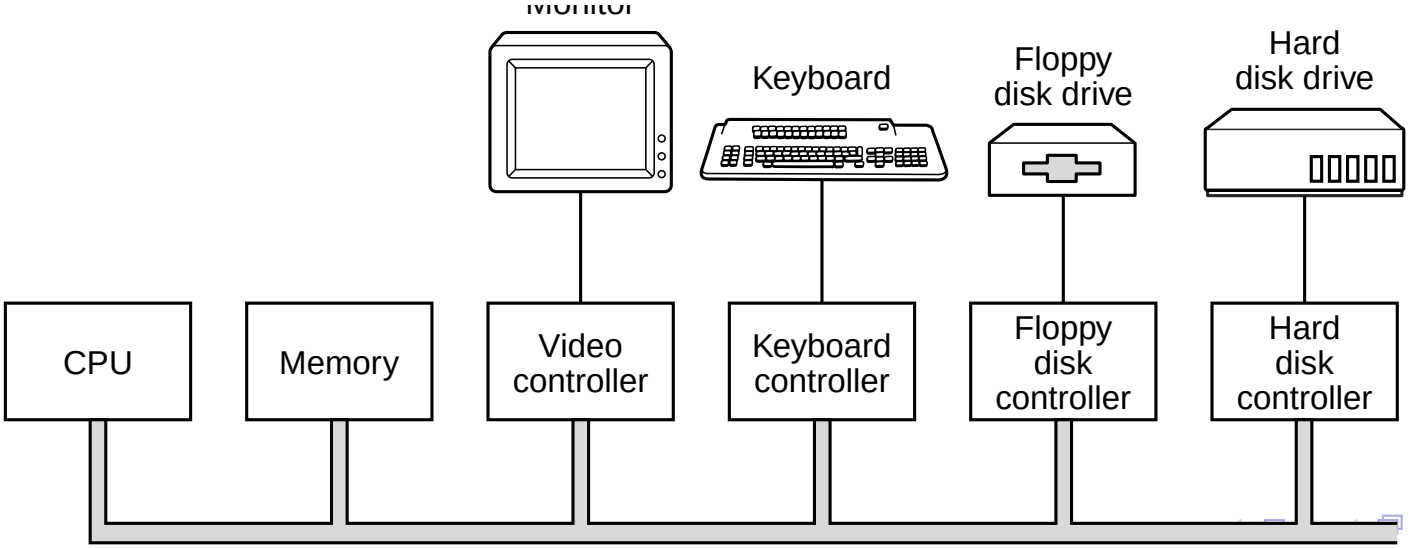
Non occorre impararle a memoria.

Mode	Description
User	programmi utente
System	subroutine certificate per l'utente
FIQ	accesso veloce a subroutine di sistema
IRQ	accesso normale a subroutine di sistema
Supervisor	livello sistema operativo
Abort	protezione memoria da parte della CPU
Undefined	usata per scopi di customizzazione.

Gestione dell'ingresso/uscita (I/O)

Risorse hardware e istruzioni macchina con cui la CPU può scambiare informazioni con le periferiche: memoria, schermo, tastiera, stampanti, rete, ...

La comunicazione è mediata da dispositivi di controllo I/O residenti nel computer, i quali permettono alla CPU di interfacciarsi in maniera uniforme con dispositivi anche molto diversi tra loro.



Accesso ai controllori della periferica

L'uniformità della comunicazione prevede la presenza di registri interni ai controllori accessibili alla CPU.

L'indirizzamento dei registri per il controllo delle periferiche può essere di tipo

- memory mapped I/O: l'accesso avviene usando un indirizzamento uniforme a quello della memoria (`LOAD`, `STORE`, ...)
- isolated I/O: l'accesso avviene usando istruzioni dedicate (`READ`, `SEND`, ...).

In ogni caso la presenza dei controllori semplifica enormemente la comunicazione con le periferiche.

Esempio: memory mapped I/O

Ogni controllore dispone di uno spazio indirizzabile dedicato, a cui accedere in lettura e/o scrittura

Dispositivo	Address range	Size
RAM I/O	0000 - 7FFF	32 KB
General purpose I/O	8000 - 80FF	256 bytes
Sound controller	9000 - 90FF	256 bytes
Video controller	A000 - A7FF	2 KB
ROM I/O	C000 - FFFF	16 KB

Al livello hardware, un circuito combinatorio abilita l'accesso ai registri nel controllore corrispondenti alle locazioni invocate.

Comunicazione con i controllori

Lo scambio d'informazione con la periferica include

- la lettura e scrittura di dati
- la scrittura di comandi
- la lettura di informazioni sullo stato.

I comandi e le informazioni sullo stato sono accessibili su registri distinti da quelli dei dati, previa segnalazione attraverso canali dedicati.

Es.: stampante

- dati da stampare
- comandi di stampa
- informazioni sull'avanzamento del lavoro, stato della stampate (mancanza di carta, livello inchiostro).

Meccanismi di sincronizzazione

La segnalazione permette di sincronizzare la comunicazione tra CPU e controllori che operano a velocità e in momenti diversi.

I meccanismi di sincronizzazione possibili sono

- I/O programmato (sincrono: busy waiting o polling)
- I/O controllato (asincrono: interrupt)
- I/O ad accesso diretto (Direct Memory Access).

I/O programmato

La CPU legge periodicamente le informazioni sullo stato di una o più periferiche all'atto di eseguire delle richieste provenienti dai programmi in esecuzione:

- semplice, in quanto non richiede hardware supplementare per l'invio di segnali alla CPU
- inefficiente, in quanto impegna periodicamente la CPU in un compito frequentemente inutile.

Il polling è usato in architetture semplici (sistemi embedded, microcontrollori).

Esempio: polling di una stampante

Es.: semplice stampante con

- segnalazione dello stato (bit `READY`)
- registro dati (carattere da stampare).

La CPU riceve un comando di stampa di una sequenza di caratteri da un programma utente e dà inizio a un ciclo di stampa tramite polling:

Esempio: polling di una stampante

Es.: semplice stampante con

- segnalazione dello stato (bit `READY`)
- registro dati (carattere da stampare).

La CPU riceve un comando di stampa di una sequenza di caratteri da un programma utente e dà inizio a un ciclo di stampa tramite polling:

1. la CPU trova la stampante pronta (`READY=1`)

Esempio: polling di una stampante

Es.: semplice stampante con

- segnalazione dello stato (bit `READY`)
- registro dati (carattere da stampare).

La CPU riceve un comando di stampa di una sequenza di caratteri da un programma utente e dà inizio a un ciclo di stampa tramite polling:

1. la CPU trova la stampante pronta (`READY=1`)
2. la CPU scrive il carattere nel registro e ne segnala l'occupazione (`READY=0`)

Esempio: polling di una stampante

Es.: semplice stampante con

- segnalazione dello stato (bit `READY`)
- registro dati (carattere da stampare).

La CPU riceve un comando di stampa di una sequenza di caratteri da un programma utente e dà inizio a un ciclo di stampa tramite polling:

1. la CPU trova la stampante pronta (`READY=1`)
2. la CPU scrive il carattere nel registro e ne segnala l'occupazione (`READY=0`)
3. il controllore legge il dato nel registro (`READY=0`)

Esempio: polling di una stampante

Es.: semplice stampante con

- segnalazione dello stato (bit `READY`)
- registro dati (carattere da stampare).

La CPU riceve un comando di stampa di una sequenza di caratteri da un programma utente e dà inizio a un ciclo di stampa tramite polling:

1. la CPU trova la stampante pronta (`READY=1`)
2. la CPU scrive il carattere nel registro e ne segnala l'occupazione (`READY=0`)
3. il controllore legge il dato nel registro (`READY=0`)
4. la stampante stampa il carattere (`READY=0`)

Esempio: polling di una stampante

Es.: semplice stampante con

- segnalazione dello stato (bit `READY`)
- registro dati (carattere da stampare).

La CPU riceve un comando di stampa di una sequenza di caratteri da un programma utente e dà inizio a un ciclo di stampa tramite polling:

1. la CPU trova la stampante pronta (`READY=1`)
2. la CPU scrive il carattere nel registro e ne segnala l'occupazione (`READY=0`)
3. il controllore legge il dato nel registro (`READY=0`)
4. la stampante stampa il carattere (`READY=0`)
5. il controllore segnala che la stampante è pronta (`READY=1`).

I/O controllato

La CPU è eccezionalmente interrotta dalle periferiche a cui ha richiesto un servizio a seguito di richieste provenienti dai programmi in esecuzione.

Quando accetta l'interruzione, la CPU passa in modalità protetta ed esegue una routine predefinita che abilita la periferica al servizio. Terminata la routine, la CPU riprende l'attività precedente mentre la periferica porta avanti il servizio.

L'I/O controllato migliora notevolmente le prestazioni, ma non risolve ancora l'obbligo da parte della CPU di gestire ogni operazione I/O, con possibilità di decadimento delle prestazioni a seguito di continue richieste d'interruzione.

Esempio: interrupt da una stampante

Es.: semplice stampante con

- segnale di interrupt (bit `PRINT_REQ`)
- registro dati (carattere da stampare).

La CPU riceve un comando di stampa di una sequenza di caratteri da un programma utente e dà inizio a un ciclo di stampa:

Esempio: interrupt da una stampante

Es.: semplice stampante con

- segnale di interrupt (bit `PRINT_REQ`)
- registro dati (carattere da stampare).

La CPU riceve un comando di stampa di una sequenza di caratteri da un programma utente e dà inizio a un ciclo di stampa:

1. la stampante chiede di stampare un carattere (`PRINT_REQ=1`)

Esempio: interrupt da una stampante

Es.: semplice stampante con

- segnale di interrupt (bit `PRINT_REQ`)
- registro dati (carattere da stampare).

La CPU riceve un comando di stampa di una sequenza di caratteri da un programma utente e dà inizio a un ciclo di stampa:

1. la stampante chiede di stampare un carattere (`PRINT_REQ=1`)
2. la CPU interrompe la sua attività ed esegue una routine di servizio (scrittura del carattere sul registro)

Esempio: interrupt da una stampante

Es.: semplice stampante con

- segnale di interrupt (bit `PRINT_REQ`)
- registro dati (carattere da stampare).

La CPU riceve un comando di stampa di una sequenza di caratteri da un programma utente e dà inizio a un ciclo di stampa:

1. la stampante chiede di stampare un carattere (`PRINT_REQ=1`)
2. la CPU interrompe la sua attività ed esegue una routine di servizio (scrittura del carattere sul registro)
3. la CPU rientra dalla routine (`PRINT_REQ=0`)

Esempio: interrupt da una stampante

Es.: semplice stampante con

- segnale di interrupt (bit `PRINT_REQ`)
- registro dati (carattere da stampare).

La CPU riceve un comando di stampa di una sequenza di caratteri da un programma utente e dà inizio a un ciclo di stampa:

1. la stampante chiede di stampare un carattere (`PRINT_REQ=1`)
2. la CPU interrompe la sua attività ed esegue una routine di servizio (scrittura del carattere sul registro)
3. la CPU rientra dalla routine (`PRINT_REQ=0`)
4. la stampante stampa il carattere (`PRINT_REQ=0`).

I/O ad accesso diretto

La comunicazione con i controllori è delegata a un dispositivo che ha accesso diretto alla memoria.

Molti vantaggi e qualche svantaggio:

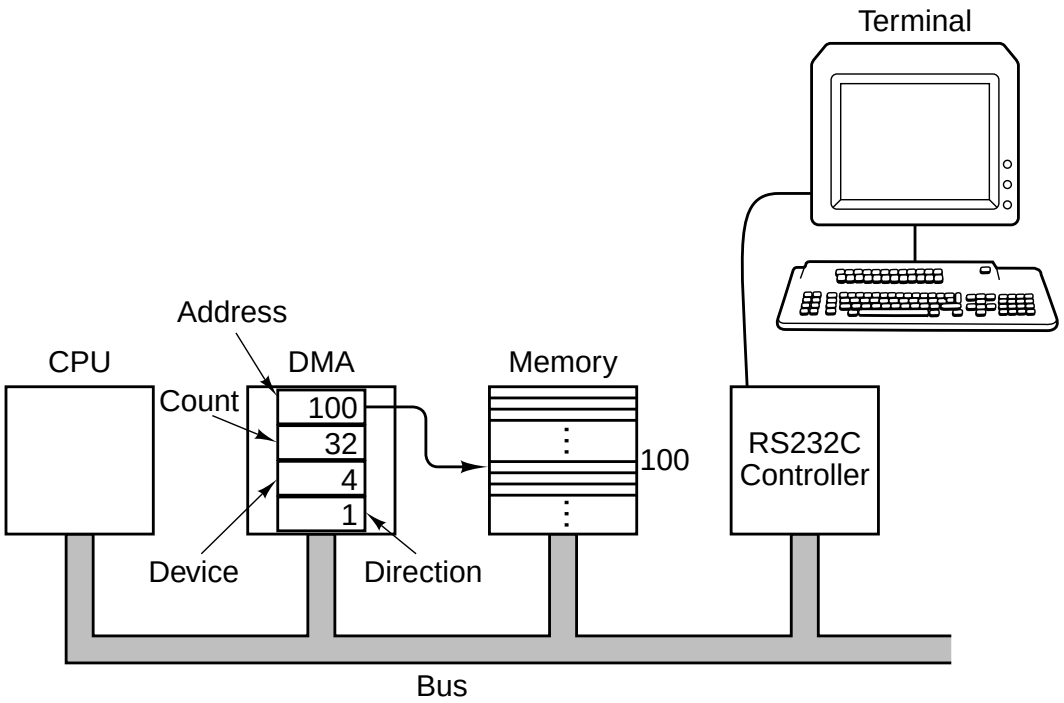
- meno interrupt da gestire per il processore
- presenza di un chip DMA aggiuntivo
- possibili conflitti per l'accesso alla memoria principale.

I controllori DMA si sono evoluti al punto di costituire un'unità locale esterna alla CPU (co-processore I/O) dotata di un proprio codice macchina per lo smistamento dei dati tra periferiche.

Esempio: DMA RS232

La CPU autorizza l'interfaccia RS232 a svolgere l'operazione, disponendo nei suoi registri

- il controllore di I/O coinvolto
- l'indirizzo della memoria principale cui accedere
- la quantità di dati da leggere o scrivere.



Procedura di gestione dell'interrupt

A seguito della generazione di un segnale di interrupt,

Procedura di gestione dell'interrupt

A seguito della generazione di un segnale di interrupt,

1. quando possibile la CPU identifica il dispositivo e accede al vettore dell'interrupt

Procedura di gestione dell'interrupt

A seguito della generazione di un segnale di interrupt,

1. quando possibile la CPU identifica il dispositivo e accede al vettore dell'interrupt
2. il vettore dell'interrupt contiene, tra l'altro, il livello di protezione e l'indirizzo della routine accessibile dalla tabella degli interrupt

Procedura di gestione dell'interrupt

A seguito della generazione di un segnale di interrupt,

1. quando possibile la CPU identifica il dispositivo e accede al vettore dell'interrupt
2. il vettore dell'interrupt contiene, tra l'altro, il livello di protezione e l'indirizzo della routine accessibile dalla tabella degli interrupt
3. la routine è avviata al livello di protezione previsto. Ciò rende la CPU non ulteriormente interrompibile se non da richieste a priorità maggiore

Procedura di gestione dell'interrupt

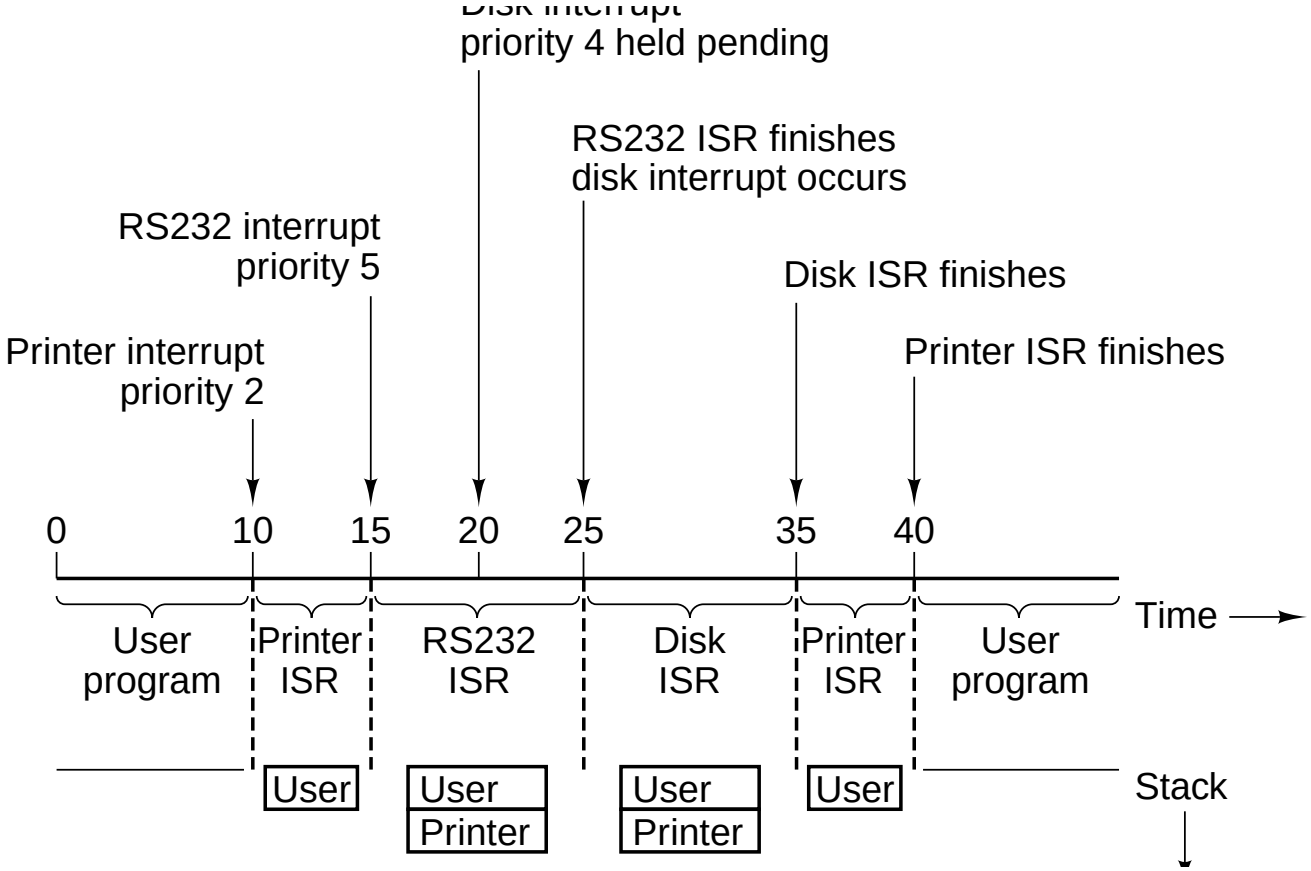
A seguito della generazione di un segnale di interrupt,

1. quando possibile la CPU identifica il dispositivo e accede al vettore dell'interrupt
2. il vettore dell'interrupt contiene, tra l'altro, il livello di protezione e l'indirizzo della routine accessibile dalla tabella degli interrupt
3. la routine è avviata al livello di protezione previsto. Ciò rende la CPU non ulteriormente interrompibile se non da richieste a priorità maggiore
4. la routine termina, la CPU rientra e prosegue l'attività precedente l'interrupt.

Mascheramento dell'interrupt

Gli interrupt che non hanno priorità sufficiente sono mascherati.

Es.: mascheramento richiesta memoria di massa



Eccezione (Trap)

Un'eccezione o trap è interrupt generato da un evento interno alla CPU stessa.

Es.: overflow della memoria, codice operativo inammissibile, indirizzo di memoria errato.

- L'interrupt esterno è sempre asincrono, in quanto indipendente dalle operazioni svolte dalla CPU.
- La trap è sempre sincrona, in quanto generata dal codice in esecuzione.

La capacità della CPU di servire le eccezioni permette di delegare alla CPU la gestione degli errori nel programma.

Precisione dell'interrupt

Un interrupt costringe la CPU a fermarsi in istanti precisi dell'esecuzione. Il parallelismo dei processori superscalari rende la gestione degli interrupt particolarmente costosa: la CPU

- deve scegliere l'istruzione su cui interrompersi
- completare le operazioni pendenti (es.: svuotare le pipeline)
- rilasciare eventuali istruzioni pre-caricate (pre-fetching).

ISA recenti hanno proposto interrupt imprecisi: non viene definita un'istruzione su cui il programma deve interrompersi. Ciò permette implementazioni più efficienti ma complica la progammazione.