1 INTRODUZIONE

1.1 Astrazioni

Per isolare i vari livelli di un computer (hardware, kernel, SO), ogni strato fornisce delle interfacce (o set di istruzioni) con la quale interagire al livello sottostante. Le più importanti sono:

- ISA (Instruction Set Architecture) → insieme istruzioni macchina
- ABI (Application Binary Interface) → interfaccia delle applicazioni

1.2 prestazioni

Le prestazioni si misurano con il **tempo di esecuzione di un programma** che è composto da 3 variabili: **numero di istruzioni**, **cicli di clock per istruzione** e **frequenza di clock**

$$\text{tempo di CPU} = \frac{numero\ istruzioni \times CPI}{frequenza\ di\ clock}$$

1.3 ISA

- influisce direttamente sul tempo di cpu (un isa è progettata per frequenze di clock più spinte, è esplicitato il numero di cicli per un'instruzione, e il numero di istruzioni stesso per compiere un'operazione) le ISA che affronteremo:

- RISC-V (RISC): cloud computing e sistemi embedded
- Intel (CISC): PC
- ARM (A-RISC): embedded e mobile

2 ARITMETICA DEI CALCOLATORI

2.1 Basi

L'aritmetica nei calcolatori viene fatta su base binaria, quindi bisogna eseguire delle conversioni da una base all'altra. per un passaggio da decimale a binario, prendo il

numero in decimale e calcolo ricorsivamente il modulo di 2, da qui tengo la parte il resto. Quando arrivo a 1, il mio numero in binario saranno i resti letti al contrario.

L'addizione in decimale e binario rimane uguale, mentre le moltiplicazioni seguono questo algoritmo: per ogni 0 del moltiplicatore mi sposto di un posto a sinistra, mentre con un 1 copio il moltiplicando. Alla fine addiziono tutto.

2.2 Codifica

2.2.1 Numeri negativi

Esistono 3 metodi per rappresentare i numeri negativi: **modulo e segno**, **complemento a 1**, **complemento a 2**. Con tutti questi metodi, il bit più a sinistra rappresenta il segno

modulo e segno: il più semplice, con il bit più a sinistra rappresenta il segno (1 per -, 0 per +)

complemento a 1: un numero positivo viene rappresento come valore assoluto, mentre uno negativo lo rappresento con complimento a 1.
I metodi del complemento sono 2:
- cambio tutti i bit da 1 a 0 e da 0 a 1
- sottraggo il numero a un numero della stessa lunghezza con tutti i bit a 1 (11111 - 00101 = 11010)

Somma e sottrazione con complemento a 1: i numeri negativi li rappresento con complemento a 1 e poi sommo i 2 numeri.Il riporto sul bit più significativo lo sommo al risultato.

6-3 $\rightarrow$ 6 + (-3) $\rightarrow$ 00110 + 11100 $\rightarrow$ 00010 con riporto di 1 $\rightarrow$ 00010 + 1 = 00011 = 3

complemento a 2: per eseguire un complemento a 2 si possono usare 3 metodi:
- metodo 1: dato un numero trovo il primo bit a 1 partendo da destra, poi eseguo il complemento a 1 per tutti i bit successivi
- metodo 2: eseguo il complemento a uno del numero, e poi addiziono 1
- metodo 3: prendo il numero con primo bit di sinistra a 1 e resto a zero e di lunghezza pari al numero che ci interessa, infine sottraggo quest'ultimo

Il complemento a 2 risulta più conveniente per le somme, la codifica dello zero è unica e nell'operazione inversa, infatto per eseguire quest'ultima si osserva il primo bit. Se è pari a 0, allora converto normalmente, se no eseguo il complemento a 2 e sommo 1, poi converto in base 10
Il vantaggio con il complemento a 2 si guadagna un valore negativo. Sfortunatamento però si posson oanche generare degli overflow. Bisogna sempre controllare

se da somma di positiva risulta in negativo o se da somma di negativi risulta un positivo

2.2.2 Rappresentazione numeri reali

virgola fissa - si dedica una parte della stringa di bit come parte intera e parte frazionaria - genericamente trattiamo il numero come intero e poi moltiplichiamo per -n (dove n rappresenta le cifre decimali) - non ci sono errori di approsimazione, ma risulta difficile gestire numeri particolarmente grandi o piccoli - per la conversione della parte decimale moltiplico per 2 e considero le parti intere

virgola mobile - un numero reale viene suddiviso come mantissa ed esponente

2.3 Codifica del testo

2.3.1 ASCII

NB: 0x denota l'utilizzo di codifica esadecimale

Per la codifica del testo si usa l'Ascii (American Standard Code for Information Interchange). Ci sono 2 versioni: ASCII e ex-ASCII (o ASCII extendend). La differenza tra i due è che il primo uso 7 bit per la codifica, mentre il secondo usa anche il bit significativo Il bit significativo del ex-ASCII, espande nuove lettere in base al codice del linguaggio (8859-1 - caratteri europa occidentale, 8859-5 - cirillico,...). L'extendend inoltre può causare problemi di condivisione data la sua dipendenza dal codice dei caratteri.
Per ulteriori caratteri si utilizza l'unicode o UTF, nelle vari versioni a 32,16 o 8 bit. l'UTF-8 è compatibile con ASCII

3 Reti logiche

- nei circuiti elettronici, i transistor hanno 2 livelli: uno alto per 1 e uno basso per 0 - le reti logiche sono circuiti che dati valori logici in entrata, ne forniscono altri un uscita - possono essere combinatorie, cioé senza memoria e l'uscita dipende solo dal valore in ingresso - possono essere sequenziali, cioé hanno memoria e l'uscita dipende anche dai precedenti ingressi - tabella di verità, espone gli uotput di tutte le combinazioni di input

3.1 Algebra di Boole

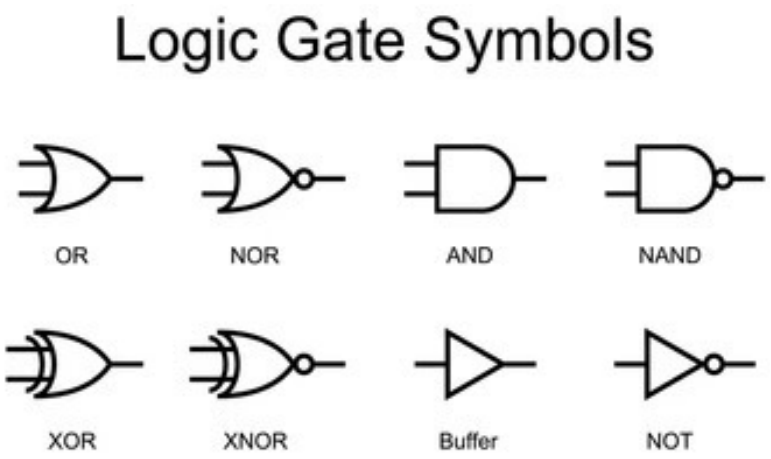
L'algebra di boole viene utilizzata per le operazioni logiche. gli operatori di base sono l'AND ($\cdot$), l'OR ($+$) e il NOT ($\overline{A}$) Come l'algebra matematica, l'algebra di boole possiede delle proprietà

- A+0=A, A·1=A (identità)
- A+1=1, A·0=0 (zero e uno)
- A+$\overline{A}$=1, A·$\overline{A}$=0 (inversa)
- A+B=B+A, A·B=B·A (commutativa)
- A+(B+C)=(A+B)+C, A·(B·C)=(A·B)·C (associativa)
- A·(B+C)=(A·B)+(A·C), A+(B·C)=(A+B)·(A+C) (distributiva)

Due di queste proprietà sono chiamate leggi di De Morgan

- $\overline{A \cdot B} = \overline{A} + \overline{B}$
- $\overline{A + B} = \overline{A} \cdot \overline{B}$

Queste ultime due proprietà introducono il NAND (NOT AND) e il NOR (NOT OR)

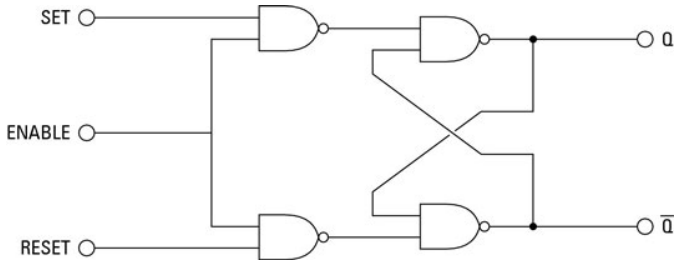


3.2 PLA (Programmable Logic Array)

La PLA è una struttura formata da un barriera di AND (mintermini) e una barriera di OR.
Le funzione logiche hanno un costo rappresentato dal numero di porte e ingressi nella rete. Il costo può essere ridotto tramite metodi di tipo sistematico o grafico e si riuniscono sotto la "sintesi logica"

3.3 Reti sequenziali

Generalmente discutiamo di reti logiche dove in un istante viene dato un output, e nell'istante successivo quell'output fa parte dei parametri in input. Ma dato che i segnali richiedono tempo per propagarsi, questo può causare errori, allora si utilizza il clock per temporizzare le azioni.
L'elemento base di memoria è chiamato **latch** (è composto da due porte NOR). I latch temporizzati vengono chiamati **gated latch**. quest'ultimi sono controllati tramite 2 AND sugli ingressi di set e reset



I latch però soffrono l'inconveniente di poter ricevere un 1 sia in set che in reset. Per risolvere il problema del doppio 1 si usufruisce dei latch-D e flip flop:
il **latch-D** riceve S e R da un unica variabile messa in NOT su R, mentre il **flip-flop** usufruisce di due latch e della negazione del clock, andando a separare nel tempo gli input

Questi elementi di memorizzazione strutturano i registri, cioé una serie di latch in grado di memorizzare una **word di dati**. Il loro funzionamento avviene al fronte in salita del clock. Questi funzionamenti in base al clock vengono definiti come **edge triggered**, e offrono il vantaggio di rimuovere le situazioni di corse.

4 Assembly

un programma in assembly è composto da una lista di istruzioni sequenziali con salti. Queste istruzioni si riuniscono in 3 macrogruppi:

- fetch: preleva un'istruzione dalla memoria
- decode: decodifica l'istruzione
- execute: eseguisce l'istruzione

Le istruzioni vengono anche categorizzate come:

- operazioni aritmetico logiche
- movimenti di dati/assegnazioni di valore
- controlli di flusso

I dati con la quale lavoriamo sono classificati come **immediati** (constanti), **contenuti in registri** (general purpose/specializzato, da 4 a 64) e **contenuti in memoria**

Infine, ci sono due tipi di architettura per la gestione dell processore: il **CISC** e il **RISC**

4.1 RISC (Reduced Instruction Set Computer)

Il RISC è un'architetture volta alla semplificazione dell'implementazione della cpu. Degli esempi di quest' archittettura sono intel e RISC-V

I comandi per le istruzioni aritmentico-logiche segue il formato $\langle opcode \rangle$ $\langle dst \rangle, \langle arg1 \rangle, \langle arg2 \rangle$ Mentre per l'accesso alla memoria la sintassi è:

- load $\langle reg \rangle$, $\langle mem\ loc \rangle$
- store $\langle mem\ loc \rangle$, $\langle reg \rangle$

4.2 CISC (Complex Instruction Set Computer)

Quest'architettura si concentra nel semplificare la scrittura dei programmi da parte del programmatore. Il numero di istruzioni è maggiore, le istruzioni aritmetico logiche hanno operandi e destinazioni in memoria, e la sintassi risulta meno regolare. Un esempio di questa architettura è ARM

5 Assembly Risc-v

Adesso esamineremo l'IS (Instruction Set) di RISC-V, un'architettura moderna e open source

5.1 istruzioni aritmentico logiche

principio di progettazione n.1:

la semplicità favorisce la regolarità

RISC-V prevede soltato istruzioni aritmentiche a 3 operandi:

commenti con #

a = b + c + d

⇓

```
add a, b, c
add a, a, d
```

Le istruzioni più complicate vanno suddivise in comandi più semplici:

a = (b+c)-(d+e)

⇓

```
add t1, b, c
add t2, d, e
sub a, t1, t2
```

In RISC-V però, gli operandi sono vincolati ad essere registri

5.1.1 Registri

principio di progettazione n.2:

minori sono le dimensioni, maggiore la velocità

Operare solo tra registri semplifica e velocizza il progetto dell'hardware. Risc-v contiene **32 registri a 64 bit**, in maniera da ridurre la propagazione dei segnali all'interno del processore. Quindi, correggendo l'esempio di prima:

a = (b+c)-(d+e)

⇓

```
add x5, x20, x21
add x6, x22, x23
sub x19, x5, x6
```

dove a = x19, b = x20, c = x21, d = x22, e = x23, t1 = x5 e t2 = x6.

5.2 La memoria

Ovviamente però il numero di registri non basta, per questo vengono usate istruzioni di trasferimento dai registri alla memoria (**store**) e dalla memoria ai registri (**load**)

Il caricamento e scaricamento di dati nei registri viene definito register spilling

Le costanti vengono memorizzate in un indirizzo. Questo metodo però risulta inefficente, quindi vengono usati gli operandi immediati:
f = f + 4 =¿ addi x22, x22, 4

5.3 I numeri

I numeri in Risc-V vengono rappresentati in esadecimale, in questo modo due cifre rappresentano un byte, e una word con 8 cifre (es: 0xEA01BD1C). Una word viene quindi caratterizzata dall'ordine di lettura dei byte più o meno significativi. Se si parte dal byte meno significativo e si sale la denoteremo little endian, se invece si parte da byte più significativo e si segnade, parliamo di big endian. Il little endian viene utilizzato in intel e RISC-V, mentre il big endian da motorola e protocolli internet RISC-V dedica un registro ad hoc (x0) alla costante 0

Le istruzioni in Risc-V vengono convertite in codici numerici univoci a 32 bit

- primo (7bit), quarto (3bit) e sesto (7bit) codificano l'istruzione
- secondo (5 bit) : secondo operando
- terzo (5 bit): primo operando
- quinto (5 bit): risultato

funz7 | rs2 | rs1 | funz3 | rd | codop

codop: codice operativo dell'istruzione funz7 efunz3: codici operativi aggiuntivi rs1: primo operando sorgente rs2: secondo operando sorgente rd: operando destinazione

Trade-off

principio di progettazione n.3:

un buon progetto richiede buoni compromessi

RISC-V con le sue istruzioni a 32 bit limita il numero di istruzioni, di registri e di modalità d'indirizzamento, ma ci guadagna in efficenza

5.4 Operazioni logiche

RISC-V offre varie funzioni per operare su porzioni di words o su singolo bit:

Shift logico
si usa per inserire degli zeri nella posizione meno (shift a sinistra) o più (shift a destra)significativa Lo shift a destra però, può generare degli overflow. Per risolvere questo problema si utilizza lo shift aritmentico (srai) che, invece di aggiungere bit pari a 0, si aggiungono bit pari al bit di segno

altre operazioni logiche sono:

AND bit a bit
la word salvata tiene valori a 1 solamente se erano a 1 in entrambe le word confrontate

OR bit a bit
la word salvata tiene il valore a 1 se compare almeno una volta nelle word confrontate

OR esclusivo (XOR)
mantiene il bit a 1 solamente se i due bit confrontati sono differenti. Viene, per esempio, usato per resettare un registro (xor x9, x9, x9)

NOT
il not è un'operatore unario e di conseguenza non è supportato da RISC-V, ma si può ottenerlo attraverso un XOR: NOT A → XOR (A, 1)

istruzioni con condizioni
nelle architetture le condizioni si convertono in istruzioni di salto condizionato. Queste istruzioni si caratterizzano in due tipi:

- per uguaglianza: beq rs1, rs2, L1 → se rs1 è uguale a rs2 allora salta all'etichetta L1
- per differenza: bne rs1, rs2, L1 → se rs1 è diverso a rs2 allora salta all'etichetta L1

es:
costrutto if: if(i==j) f=g+h; else f=g-h;

```
        bne x22, x23, ELSE
        add x19, x20, x21
        beq x0, x0, ESCI
ELSE:   sub x19, x20, x21
ESCI:
```

Cicli sfruttando i salti condizionati si possono creare dei cicli, per esempio:

```
Ciclo:  slli x10, x22, 3
        add x10, x10, x25
        ld x9, 0(x10)
        bne x9, x24, Esci
        addi x22, x22, 1
        beq x0, x0, Ciclo
Esci:   ...
```

Alcune considerazioni:
le istruzioni tra 2 salti condizionati (conditional branch) vengono chiamati blocchi di base. L'individuazione di questi blocchi è una delle prime fasi della compilazione.

ulteriori istruzioni di salto sono *blt*(salta se minore),bge (salta se maggiore o uguale), bltu e bgeu (blt e bge ma unsigned). Le versioni unsigned semplicemente tengono conto o meno del bit significativo

costrutto switch
Memorizza i vari codici da eseguire in una tabella, per poi caricare in un registro l'indirizzo del codice da eseguire.

infine abbiamo l'istruzione di salto jalr che salta all'indirizzo contenuto nel registro

5.5 Procedure

Le funzioni (o procedure) svolgono un ruolo fondamentale,ma necessitiamo di un protocollo per richiamarle. Un protocollo necessità di standardizzare

1. dei posti noti dove caricare i parametri in input
2. il trasferimento del controllo alla procedura

   aquisizione dei parametri in input

   esecuzione della procedura

   caricamento dei valori di ritorno in posti noti

   restituzione del controllo al chiamante
3. la raccolta dei valori di ritorno e la pulizia delle tracce della procedura

RISC-V affronta questo meccanismo usando il più possibile i registri attraverso delle convenzioni:

- x10-x17: usato per parametri in ingresso e valori di ritorno
- x1: indirizzo di ritorno

  questo indirizzo viene usato particolarmente per istruzioni di jump and link (jal), che effettuano il salto e memorizzano in x1 l'indirizzo di ritorno

  alla fine della procedura sarà sufficiente fare un salto (jalr x0, 0(x1))

nel caso i registri non bastino viene usato lo stack x2, chiamato anche **sp** (Stack Pointer). Alla fine della procedura è importante ricordare di pulire lo stack

- x5-x7 e x28-x31 sono registri temporanei
- x8-x9 e x18-x27 sono registri in cui salvare il contenuto in caso di chiamata a procedura

In caso di variabili locali e procedure annidate la gestione dei registri può diventare complessa Questo viene risolto usando lo stack per allocare variabili locali e valori del registro di ritorno x1

Storage class le variabili in C sono associate a locazioni di memoria caratterizzate per tipo e storage class. Il C ha due tipi di storage class: Automatic (le variabili locali) e Static. Le variabili static vengono memorizzate da RISC-V nel registro $x3$, chiamato anche **gp** (global pointer). Per quello che riguarda le automatic, esse sono memorizzate nei registri o nello stack in caso i registri non bastassero

record di attivazione Il segmento di stack che contiene variabili locali e i registri salvati è chiamato record di attivazione (o stack frame). Le variabili locali sono individuate tramite un offset. In alcuni casi il registro sp può essere scomodo, percui in alcuni programmi viene usato il registro x8 come **fp** o frame pointer, come puntatore alla prima parola doppia del frame della procedura

per tutti le variabili allocate dinamicamente, RISC-V procede per indirizzi crescenti

Riassunto convenzione sui registri:

Nome	Numero del registro	Utilizzo	Da conservare nella chiamata?
x0	0	Costante 0	n.a.
x1 (ra)	1	Registro di ritorno (registro di collegamento)	sì
x2 (sp)	2	Stack pointer	sì
x3 (gp)	3	Global pointer	sì
X4 (tp)	4	Thread pointer	sì
x5–x7	5-7	Variabili temporanee	no
x8–x9	8-9	Variabili da preservare	sì
x10–x17	10-17	Argomenti/risultati	no
X18–x27	18-27	Variabili da preservare	sì
x28–x31	28-31	Variabili temporanee	no

6 Intel

Intel è un'architettura di tipo CISC usata in laptop, desktop e servers. Possiede diverse modalità di funzionamento e diversi assembler. Nel nostro caso useremo GNU

6.1 Registri

Intel usa 16 registri general purpouse a 64 bit caratterizzati dal prefisso %. Alcuni dei più importanti sono:

- %rsp: stack pointer
- %sbp: base pointer
- %rsi e %rdi: registri source e destination per la copia di Array
- %rip: Instruction pointer
- %rflags: estende %flags, che contiene flag come CF (flag di carry), ZF(risultato 0), SF(risultato negativo), OF(flag di overflow). Questi registri speciali vengono usati da istruzioni di salto condizionale

6.2 Convenzioni di chiamata

Le chiamate in Intel sono molteplici, e cambiano in base alle specifiche tecniche dell'ABI. Alcune però sono comuni a tutte le CPU:

- i primi 6 argomenti sono in %rdi, %rsi, %rdx, %rcx, %r8 ed %r9, mentre gli altri vengono impilati nello stack
- i valori di ritorno stanno in %rax e %rdx
- i registri vengono preservati %rbp, %rbx, %r12, %r13, %r14 ed %r15
- i valori di ritorno in %rax e %rdx

6.3 Indirizzamento

Con intel le istruzioni sono principalmente a 2 operandi con destinazione implicita. Per la sorgente si usano operandi immediati($20), registri (%rax), o indirizzi di memoria (0x0100A8). Per le destinazioni sono utilizzabili. L'accesso alla memoria avviene seguendo la seguente sintassi:

$\langle displ \rangle$ ($\langle basereg \rangle, \langle indexreg \rangle, \langle scale \rangle$)

- $\langle displacement \rangle$: costante a 8, 16 o 32bit
- $\langle base \rangle$: valore in registro
- $\langle indice \rangle$: valore in registro
- $\langle scala \rangle$: valore costante

In casi speciali scala, indice, base e displacement possono essere omessi

6.4 Istruzioni intel

Le istruzioni Intel sono molte, inutile sarebbe elencarle tutte, ma condividono una sintassi comune:

$\langle opcode \rangle \langle source \rangle, \langle destination \rangle$

Il parametro $\langle opcode \rangle$ può terminare con b/w/l/q per indicare che l'operazione viene effettuata su 8/16/32/64bit

istruzioni comuni

- istruzioni logico aritmentiche

  mov, add, sub, mul, div, inc/dec (incrementa/decrementa di 1),
  rcl/rcr/rol/ror (rotate)
  sal/sar/shl/shr (shift aritmentici e logici)
  and, or, xor, not, neg(complemento a 2, negazione), nop.

- push/pop: inserisce/rimuove dati nello stack
- cmp/test: comparano i 2 argomenti e settano i flag nel flag register
- jmp, je/jnz/jc/jnc: salti condizionati
- call/ret: chiamata/ritorno alla procedura
- lea (Load effective address) è un istruzione utilizzata per calcolare indirizzi:

  calcola e e memorizza l'indirizzo senza caricare nulla dalla memoria
  viene anche usato per fare somme

7 ARM

- Advanced Risc Machine - usato nei sistemi embedded e in smartphone e tablet - 16 registri e modalità d'indirizzamento potenti - precisazioni, ARM è una famiglia di CPU, con ISA differenti ma simili fra loro

registri - 16 registri a 32 bit, r0-r15, quasi tutti general purpouse (r15 non lo è) - alcuni registri hanno sinonimi: r13=sp(stack pointer) e r14=lr(link register) - Application Program Status Register (apsr) / Current Program StatusRegister (cpsr) ? - r15 contiene il pc(program counter) e i flags (bit 28..31) - flags: z — c — n — v - eq/ne: equal/not equal =¿ z = 1/0 - hs/lo: higher or same/lower =¿ c = 1/0 - mi/pl: negativo o positivo =¿ n = 1/0 - vs/vs: presenza o meno di overflow =¿ v = 1/0 - hi: higher =¿ c = 1 — z = 0 - ls: lower or same =¿ c = 0 oppure z = 1 - ge: greater or equal =¿ n = 1 — v = 1 oppure n = 0 — v = 0 - lt: less than =¿ n = 1 — v = 0 oppure n = 0 — v = 1 - gt: greater than =¿ come ge ma con z = 0 - le: less or equal =¿ come lt, ma esegue anche se z = 1

Convenzioni di chiamata - r0-r3 sono registri temporanei non preservati, con più di 4 argomenti si usa lo stack - r4-r11 registri preservati (in alcune ABI r9 non è preservato) - r0-r1 valori di ritorno

Indirizzamento - istruzioni prevalentemente a 3 argomenti - operando sinistro = registro, operando destro = immediato o registro - accesso alla memoria con ldr/str (load register/store register) e ldm/stm (load/store multiple)

8 Toolchain

La CPU funziona con un linguaggio di basso livello composto, da 1 e 0, troppo complesso da programmare. La conversione viene eseguita dal compilatore

8.1 Esempio: il compilatore di C

la compilazione del linguaggio C segue i seguenti passaggi:

1. Preprocessore: gestisce le direttive di sostituzione del codice
2. Compilatore: da C ad assembly
3. Assembler: da assembly a linguaggio macchina
4. Linker: unisce linguaggio macchina con le librerie per creare un eseguibile

Questo processo è gestito automaticamente da un Driver

gcc (Gnu Compiler Collection)
È in grado di compilare vari linguaggi in assembly per varie CPU. Per operare utilizza vari programmi: cpp,cc,as e ld

Da C ad Assembly
dato un file .c, il comando gcc -S invoca cc per generare un file assembly .s. cc riconosce l'architettura dulla CPU, applica vari livelli di ottimizzazione (tramite l'opzione -o)

Da Assembly a linguaggio macchina
Usando gcc -c viene invocato cc e successivamente as, per generare un file .o, o file oggetto. as non solo converte i codici memonici in sequenze di bit, ma converte pseudo-istruzioni, i numeri da decimale/esadecimale a binario, genera metadati e gestisce label e salti.
Con pseudo-istruzioni si intendo le istruzioni non di tipo nativo, che sono formate da altre istruzioni macchina e che esistono per semplificare la programmazione.

File oggetto
È composto da segmenti distinti:

- header: specifica dimensione e posizione degli altri segmenti del file oggetto
- text segment: contiene il codice in linguaggio macchina
- data segment: contiene tutti i dati statici e dinamici allocati
- symbol table: associa simboli a indirizzi e enumera quelli non definiti
- relocation table: enumera istruzioni e dati che dipendono da indirizzi assoluti, o definiti quando il programma viene caricato in memoria
- altro...

Da file oggetto a eseguibile
Il passo finale, tramite gcc senza opzioni, richiama il linker ld, che decide la disposizione di codice e dati nella memoria, associa indirizzi assoluti a tutti i simboli, risolve i siboli lasciati indefiniti nel file .o e sistema la tabella di rilocazione. In breve, ld rimuove le tabelle dei simboli e di rilocazione, generando codice macchina con riferimenti corretti.
Un linker gestisce vari tipi di simboli:

- simboli definiti: associati a un indirizzo nella tabella dei simboli
- simboli non definiti: usati in un file, ma definiti in un file diverso. Se il linker, cercando in altri file non trova il simbolo, da errore di linking
- simboli locali: definiti e usati in un file, ma non usabile in altri file

in tre passi, il linking consiste nel: disporre in memoria i vari segmenti dei file .o, assegnare un indirizzo assoluto ad ogni simbolo contenuto nelle tabelle e correggere le istruzioni delle tabelle di rilocazione con degli indirizzi calcolati. Il risultato viene poi incapsulato in un file eseguibile (segmenti, informazioni per il carimento in memoria, altri dati)

8.2 Librerie

il compilatore, o il sistema, fornisce delle funzioni predefinite. Queste funzioni sono contenute in librerie, o collezioni di file .o. Le librerie sono di due tipi: statiche (.a) e dinamiche (.so)

Librerie statiche
ld inserisce tutto il codice della libreria nell'eseguibile. Quest'ultima serve solo durante il linking.

Librerie dinamiche
ld inserisce un riferimento a un linker dinamico (es: /lib/ld-linux.so) alla libreria, senza includerla nell'eseguibile. All'esecuzione del programma, il linker viene caricato ed eseguito, caricando le librerie e l'eseguibile, facendo il linking.

La differenza principale tra librerie statiche e dinamiche sta nelle dimensioni e nella complessità di caricamento del programma da parte del SO. Un'altro vantaggio delle librerie dinamiche rispetto a quelle statiche è che dato che le librerie non sono caricate, permette di aggiornarle senza dover ricompilare l'eseguibile.

Lazy linking
Può capitare che alcune librerie vengano poco usate a tempo di runtime, quindi, invece che linkare la vera funzione, può essere chiamato uno stub, che esegue caricamento, rilocazione e linking solo quando necessario

(es: slide 18, toolchain)

9 Il processore

panoramica - nell'esecuzione delle istruzioni, ci sono due fasi iniziali comuni - prelievo dell'istruzione dalla memoria - lettura dei registri - i passi sccessivi sono simili, ma dipendono dall'istruzione

-le istruzioni più comuni usano la alu - calcolo dell'indirizzo per l'accesso alla memoria - istruzioni aritmentico logiche - confrontare le condizioni per eseguire dei salti - successivamente le tre classi divergono - le istruzioni d'accesso alla memoria salvano il dato in memoria - le istruzioni aritmetico logiche memorizzano il risultato nel registro - le istruzioni di salto cambiano il valore del registro PC in base al confronto

Il processore utilizza anche i multiplexer per effettuare decisioni. Per esempio le istruzioni possono provenire dal banco dei registri o dal codice dell'istruzione stessa, per selezionare i due tipi, viene usato un multiplexer. Oppure per far decidere alla ALU che operazione effettuare, dai banchi dei registri per decidere in quale registro scrivere o dalla memoria dati per determinare se vogliamo leggere o scrivere. Per decidere come utilizzare i vari ingressi di controllo viene utilizzata un'unità specifica.

figura slide 10

9.1 Reti logiche

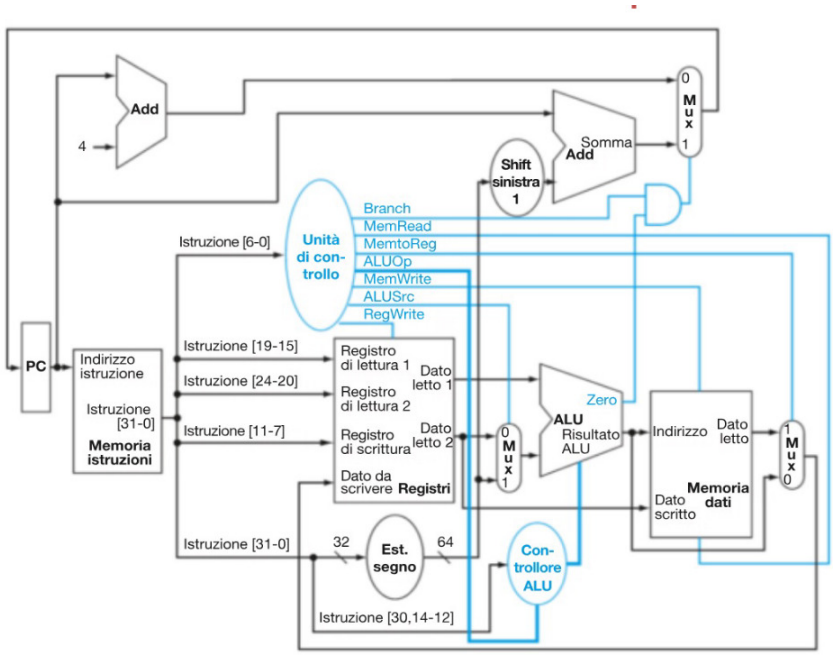
una rete logica combinatoria consiste in un circuito di porte logiche dove l'output è una funzione statica dell'input. Contiene anche elementi di stato, cioé registri, memorie dati e memorie d'istruzioni. Questi elementi di stato sono detti sequenziali perché l'uscita a un ingresso dipende dagli ingressi precendenti.

Flip-flop
L'elemento base per la memorizzazione dei bit è il flip-flop D-latch. Un array di latch forma Un registro. Un latch viene definito come asserito o non asserito in base al livello logico che possiede

9.2 Temporizzazione

La temporizzazione è la metodologia che determina quando i segnali possono essere letti o scritti in base al ciclo di clock. La più utilizzata è quella dei fronti, cioé della salita o discesa dei fronti di clock.

9.3 Data path



Memoria istruzioni
memoria che contiene le istruzioni

Program Counter
Registro che contiene l'indirizzo dell'istruzione da caricare

Sommatore (ALU)
ALU specializzata per incrementare in PC

ALU
Effettua calcoli e confronti, a un parametro di uscita speciale che viene impostato a 1 quando da un confronto risulta che i due elementi sono uguali

Controllore ALU
La ALU viene controllata tramite dei bit di controllo generati da questa unita. Questi bit sono determinati dai campi funz3, funz7 e dai due bit ALUop. I bit ALUop sono definiti dall'unità di controllo

Banco dei registri
prende in entrata i registri dove leggere/scrivere e il dato da scrivere, e in uscita rida i dati letti

Unità di memoria dati
Questa memoria si differisce dalla memoria istruzioni per la capacità di operare sia in lettura che in scrittura

Unità di estensione del segno
viene utilizzata per estendere il campo a 12bit a 64bit per applicare l'offset usato nei salti condizionali

RegWrite
Se abilitato scrive nel registro specificato il dato in ingresso

ALUSrc
Applica un'estensione del segno al secondo operando della ALU

PCSrc
Applica un'estensione del segno per effettuare dei salti nelle istruzioni contenute nel PCSrc

MemRead/MemWrite
Determina il tipo di interazione con la memoria dati

MemtoReg
Determina se il dato inviato al register file provenga dalla ALU o dalla memoria

Branch
Interagisce con il segnale di zero proveniente dalla ALU per effettuare un salto a istruzione

Unità di controllo
È una componente che genera tutti i segnali di controllo in base al codice operativo dell'istruzione ricevuta da PC. I segnali con la quale interagisce sono ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, ALUOp1, ALUOp0

10 Pipeline

Il ciclo di clock è definito dalle istruzioni più lente, per esempio quelle di accesso alla memoria. Per ottimizzare i tempi di inattivita vengono implementate le pipelines, che permettono la paralelizzazione dei processi. Generalmente il miglioramente viene calcolato come:

$$tempo\ tra\ due\ istruzioni\ con\ pipeline\ =\ \frac{Tempo\ senza\ pipeline}{Numero\ di\ stadi}$$

Dove con stati intendiamo le diverse fasi che compongono un'istruzione.
In realtà questa regola non si può applicare perfettamente a causa della durata di alcune istruzione. In generale il miglioramento è inferiore rispetto a quello calcolato tramite la formula.

10.1 Pipelines e architetture RISC

L'architettura RISC offre dei vantaggi specifici che migliorano l'implementazione delle pipelines:

- Tutte le istruzioni hanno la stessa lunghezza, con conseguente prelievo più efficente
- i codici degli operandi sono in posizione fissa. Questo permette la lettura del register file in parallelo con la decodifica dell'istruzione
- gli operandi in memoria sono disponibili solo per ld e sd. Questo permette il calcolo degli indirizzi in ALU
- gli accessi allineati permettono un solo ciclo di trasferimento per gli accessi in memoria

10.2 Hazard

A causa del loro funzionamento, alcune istruzioni non possono essere eseguiti in un solo ciclo di clock. Le condizioni che si generano vengono definite *hazard* e possono essere di diverse tipologie

10.2.1 Hazard strutturali

L'hazard strutturale è una condizione dove l'architettura stessa del processore rende impossibile l'esecuzione di alcune istruzioni in pipeline. Per esempio, con una sola memoria non è possibile sia caricare che prelevare operandi nello stesso ciclo di clock.

10.2.2 Hazard sui dati

Questa condizione si verifica quando un'istruzione necessità di dati provenienti da un'istruzione precedente. quindi il processore deve fermare l'istruzione ed attendere finche il necessario dato non viene salvato nell'istruzione precedente. In alcuni casi il compilatore è in grado di effettuare alcune ottimizzazioni per ottenere quel risultato. Una di queste ottimizzazioni è *l'operation forwarding*. Nel caso abbiamo due istruzioni algebriche nel quale il risultato della prima è operando della seconda, l'operation forwarding salta la parte di scrittura nel registro del risultato, e lo reinserisce direttamente nella ALU.

10.2.3 Hazard sul controllo

Quando viene effettuato un salto condizionato, c'è bisogno di attendere il risultato del controllo (o della condizione del salto) per determinare qual'è l'istruzione successiva da caricare. Questo tempo di attesa è un hazard di controllo. Maggiore è il numero di stadi di un'istruzione, maggiore è il ritardo sulla pipeline.
Per risolvere questo hazard vengono usufruiti vari metodi per la previsione del salto. Uno di questi consiste nel caricamento dell'istruzione successiva, e nel caso le condizioni di salto siano positive, l'istruzione viene abortita dalla pipeline e sostituita con quelle espressa nel salto. Questo non è il metodo più efficente, infatti esistono circuiterie più complesse che permettono di memorizzare l'esito del branch e mantenere un comportamento coerente

11 Gerarchia di memoria

11.1 Terminologia

Memoria indirizzata direttamente
Di tipo volatile, è limitata dallo spazio di indirizzamento del processore e accessibile in qualsiasi momento nella memoria principale

Memoria indirizzata indirettamente
Di tipo permanete, possiede uno spazio di indirizzamento di tipo software. Generalmente è il sistema operativo a gestire il passaggio tra memoria principale e periferica

Tempo di accesso
Tempo richiesto per una operazione in lettura o scrittura

Tempo di ciclo
Il tempo dell'intervallo tra due operazioni consecutive

Accesso casuale
I dati non sono ne in ordine ne condividono alcuna relazione. Accesso tipico di memorie a semiconduttori

Accesso sequenziale
L'accesso alla memoria è ordinato o semi-ordinato. Il tempo d'accesso dipende dalla posizione

RAM (Random Access Memory)
Memoria in lettura/scrittura a semiconduttori

ROM (Read Only Memory)
Memoria a semiconduttori in sola lettura

11.2 Tipi di memoria

11.2.1 Memorie RAM

Le RAM memorizzano singoli bit in byte e/o word. Può supportare il parallelismo della capacità (128K $\times 4$), questo influenza il numero di pin I/O

11.2.2 Memorie statiche (SRAM)

In questo tipo di memoria, i bit possono essere tenuti indefinitamente, finche è presente l'alimentazione. Sono estremamente veloci e consumano poca corrente, il tutto a un costo maggiore.

Sono formate da una linea word che determina l'appertura di due interruttori T_1 e T_2 che isolano un latch. Infine il dato viene trasmesso tramite due linee d e d^I, dove $d^I = NOT(d)$

11.2.3 RAM Dinamiche (DRAM)

Economiche e a alta densità, mantengono la carica attraverso un codensatore. Necessitano di un *refresh* continuo per mantenere il dato, il che aumenta i consumi.

È formata da un interruttore che chiude il circuito a un condensatore, da due linee di word e di bit e un *sense amplifier*, cioé un circuito che mantiene la tensione sul condensatore. Il sense aplifier viene attivato a ogni ciclo di lettura, quindi il chip di memoria contiene un circuito per una lettura periodica di tutta la memoria