Architettura degli elaboratori

Matteo

February 6, 2026

1 Introduction

Un calcolatore è un sistema composto da processsori, memorie, e dispositivi di input/output. un **Bus** è un insieme di connessioni elettriche parallele che trasportano dati da un componente all'altro.

Nell'architettura di Von Neumann viene introdotta l'idea di usare la memoria non solo per i dati ma anche per il programma.

Collega la cpu e la memoria con un bus indirizzi, indicando la posizione dei dati. Mentre su un altro bus dati memoria e cpu si scambiano i dati veri e propri.

CPU composta date

- Unita di controllo: legge e interpreta le istruzioni.
- Unita aritmetico logica (ALU): esegue operazioni aritmetiche e logiche.
- registri: memorie veloci interne alla cpu.

ci sono diversi registri specializzati:

- Program counter (PC): indica la prossima istruzione da eseguire.
- Memory address register (MAR): contiene l'indirizzo di memoria da cui leggere o scrivere dati.
- Memory data register (MDR): registro che accede ai bus dei dati.
- Instruction register (IR): contiene l'istruzione che si sta per eseguire.
- Program status word (PSW): contiene informazioni sullo stato della CPU.

in pratica il ciclo di esecuzione di un'istruzione è:

1. il contenuto di PC viene copiato in MAR e inviato alla memoria attraverso il bus indirizzi.
2. la memoria risponde inviando il dato all'MDR attraverso il bus dati.
3. il contenuto di MDR viene copiato in IR e decodificato.
4. l'istruzione passa alla ALU per essere eseguita.
5. se servono altri dati vengono letti dalla memoria nello stesso modo.

6. se serve il risultato e copiato in memoria attraverso MDR

7. il PC viene aggiornato per puntare alla prossima istruzione.

Questo ciclo e chiamato **fetch-decode-execute cycle**.

data path: insieme di registri e ALU che eseguono operazioni sui dati.

il percorso dei dati da memoria ad ALU, la loro esecuzione e il ritorno in memoria viene chiamato **ciclo di data path**, ed e governato da un clock.

durata ciclo di data path o ciclo di clock= 1/F, dove F e la frequenza di lavoro della CPU(cicli al secondo), misurata in heartz.

la velocita di esecuzione delle istruzioni ISA(instruction set architecture), è misurata con la durata di un ciclo di clock per i cicli necessari.

fissato il ciclo di clock, la velocita puo essere aumentata con il parallelismo.

negli anni 70 si sviluppo la differenza tra CISC e RISC. CISC: complex instruction set computer, set di istruzioni complesso, con istruzioni che fanno operazioni complesse in un singolo ciclo. RISC: reduced instruction set computer, set di istruzioni ridotto, con istruzioni semplici che richiedono piu cicli per operazioni complesse.

Risc usa la microprogrammazione, in modo da far fare diverse operazioni a pochi componenti.

piu cicli FDE possono essere eseguiti insieme con il **pipelining**

si sviluppano anche architetture che permettono il parallelismo. , ad esempio cpu con piu ALU e control units.

oppure molte cpu possono lavorare in modo coordinato sulla stessa istruzione su dati diversi, creando un **array computer**

se invece eseguono istruzioni diverse su dati, con una memoria condivisa, diversi si parla di **multiprocessor systems**.

la forma piu complessa sono i **multicomputer systems**, con piu cpu, ognuna con memoria propria, collegate in rete.

1.1 memoria

vari tipi di memoria

- memoria volatile: perde il contenuto quando viene spenta(es:ram).
- memoria non volatile: mantiene il contenuto quando spenta(es:rom, hard disk).
- memoria online: sempre accessibile
- memoria offline: il supporto deve essere montato

la memoria si organizza in celle, ogni cella è una sequenza di bit con il proprio indirizzo.

cella di 8 bit=byte.

molti calcolatori lavorano su blocchi da 32 o 64 bit, questi blocchi si chiamano word.

le word possono essere memorizzate in celle standard, occupando piu celle consecutive. si puo fare in 2 modi:

- big-endian: byte piu significativo all'indirizzo piu basso.
- little-endian: byte meno significativo all'indirizzo piu basso.

la **cache** è una memoria poco capiente ma veloce, che viene usata per contenere le word piu usate.

la cpu prima cerca i dati nella cache, se non li trova li va a prendere in memoria principale.

principio di località: dati usati recentemente gli uni dagli altri sono spesso in locazoni vicine.

tipi di supporti:

- hard disk: memoria non volatile, lenta, grande capacita, usa piatti magnetici rotanti.(puo essere resa piu veloce con la tecnica RAID, che usa piu dischi in parallelo).
- SSD: memoria non volatile, veloce, grande capacita, usa memoria flash, piu veloce ma meno capiente.
- CD e DVD: memoria non volatile, usano supporti ottici, capacita limitata.

2 porte logiche e circuiti combinatori

Porte logiche sono basate sull'algebra di Boole, un'espressione booleana si costruisce con: le costanti di boole(0,1), gli opeartori booleani, e variabili(x,y, ecc..).

mintermine: un prodotto (AND) di tutte le variabili della funzione, ognuna presente una sola volta, negata o non negata, che vale 1 per una sola combinazione di valori delle variabili.

forma canonica: l'OR di tutti mintermini veri

porta nand è una porta logica utile per costruire circuiti combinatori, falsa solo quando a=1 e b=1. è universale e facile da costruire con transistor

array logici programmmabili: collegando dei fusibili è possibile creare porte logiche personalizzate.

2.1 mappe di karnaugh

mappe di karnaugh: modo di rappresentare funzioni booleane. permettono di costruire un circuito combinatorio minimale.

tabella bidimensionale in cui ogni mintermine ha una cella, e la cella ha un valore, 1 se vale, se 0 se no.

puoi creare raggruppamenti che comprendono gruppi di certi letterali =1.

all'interno di una mappa di karnaugh una copertura si dice minimale quando:

- i suoi raggruppamenti non è contenuto in raggruppamenti piu grandi.
- i suoi raggruppamenti contengono almeno una cella che non appare in altri raggruppamenti della copertura

2.2 bus multi bit

circuiti combinatori collegati a connessioni da piu bit sono detti **bus**

sono praticamente un raggruppamento di segnali e ogni coppia lavora separatamente come in una porta normale, per poi far uscire i seganli uno dopo l'altro.

3 codice binario

codifiche posizionali principali:

- binario
- ottale
- esadecimale

per convertire da bianrio a ottale prendi 3 cifre alla volta e calcola il numero, per esadecimale prendine 4.

i numeri possono anche essere codificati **in eccesso** cioe dato k= bits e bias= 2^{k-1}. somma numero e bias, e codifica regolarmente il risultato.

es: -127 + bias(128) = 1. 1= 00000001

se poi vuoi decodificare, decodifica il binario nel numero e sottrai il bias.

nelle somme tra binari, il riporto finale nel complemento a 1 viene sommato all'inizio, nel complemento a 2 viene scartato.

puo verificarsi **overflow**: se i due addendi hanno segni diversi non accade mai, se hanno lo stesso segno e il risultato ha segno opposto, si verifica overflow.

3.1 codifica a virgola mobile

composto da sue parti: **mantissa**(o frazione): numero, ed **esponente**: potenza

le operazioni su numeri floating point(virgola mobile) possono dare 2 tipi di errore

- overflow: numero in valore assoluto troppo grande
- underflow: numero in valore assoluto troppo piccolo

normalizzare un numero significa scrivere un numero in una forma standard dove la mantissa ha sempre la stessa struttura.

in binario vuoi fare in modo che il numero piu significativo della mantissa(il primo) sia 1.

puoi farlo spostando il primo 1 a sinistra di x posti e poi sottraendo x all'esponente e ricodificandolo.

3.2 standard di codifica

IEEE 754 è lo standard piu usato per i floating point. definisce diversi formati tra cui Binary32 che usa 1 bit di segno, 8 di esponente e 23 di mantissa.

era impossibile codificare caratteri extraeuropei in ASCII quindi fu creato UNICODE, che usava 16 bit. UNICODE pero usa troppi bit per i caratteri piu semplici ed e vicino oramai all'esaurimento, quindi fu creato UTF-8. UTF-8 puo dinamicamente occupare da 1 a 4 byte, I bit iniziali indicano il formato specifico e la quantità di bit utilizzati.

3.3 bit di controllo

i codici binari o in lettura o in scrittura possono essere soggette a errori, per evitare questo esistono i bit di controllo.

esistono sia bit di rilevazione che di correzione.

distanza di Hamming: è il numero di bit di differenza tra due parole. la distanza di hamming di un codice è invece la minima dist. di Hamming tra tutte le parole del codice.

per rilevare d bit errati servono, la distanza di Hamming deve essere uguale a d+1, per correggerli deve essere d+2.

uno dei codici di controllo piu semplici e il **bit di parita**: un bit che fa in modo che il numero di 1 sia pari, la dist. di hamming minima risulta essere 2.

i bit di parita devono sempre rispettare l'equazione m(numero bit)+ r(num bit di contollo)+1 minore o uguale a 2^r e vanno collocati nelle posizioni equivalenti alle potenze di 2 (1,2,4,8,16 ....) e ognuno controlla un sottogruppo di bit.

4 circuiti sequenziali

alcuni circuiti possono memorizzare l'input in e ripeterlo, in questi casi l'output non dipende piu solo dall'input, ma anche da altre circostanze.

un esempio e il latch SR. il circuito contiene un ciclo e se entrambi gli input sono 0 il risultato e sconosiuto.

si puo fare in modo che il latch risponda agli input solo durante certi momenti mettendo gli input e il ciclo di clock in porte AND

il latch D ha un solo input e cio fa in modo che non si verifichi la situazion in cui entrambi gli input sono 1.

Il latch D cambia il proprio stato mentre il clock è attivo, durante la fase 0 del clock memorizza l'ultimo valore di D alla fine della fase 1

i circuiti con **commutazione sul fronte**: ovvero due stati stabili che cambiano in base al clock, sono detti flip-flop

esistono vari flip-flop, noi usiamo il tipo D.

il DFF(flip-flop tipo D) cambia stato dopo il fronte di salita in base in base agli input che ha ricevuto durante il fronte di salita.

gli **one-bit-register**: memorizzano un bit nell'ultimo istante in cui load e stato attivato

collegando tanti one-bit-registers puoi creare un w-bit-register

esiste anche il **counter**: e un fcircuito sequenziale usato per crare i program counter, oltre a load ha reset che resetta a 0 il bit interno e inc che aumenta di 1 il valore memorizzato. se nessuno di questi comandi e usato ripete solo il val memorizzato.

una memoria con locazioni da w bit puo essere realizzata con w-bit-registers e un circuito chiamato DIrect Access Logic indica a quale accedere. cosi si realizzano registri di memoria.

puo essere utili avere circuiti logici anche dopo le uscite delle memorie per indicare quale delle uscite dei registri deve essere portata su out.

5 microarchitettura

il livello della microarchitettura si occupa di utilizzare le parti del livello logico digitale per creare linguaggio macchina.

il processore hack ad esempio ha due registri D e d A che possono dare alla ALU i suoi 2 input,D puo contenere un precedente output, A un'istruzione o un precedente output.

il flusso dei dati tra componenti viene gestito con MUX.

i MUX e i bit di controllo vengono gestiti da una microarchitettura di circuiti combinatori. L'intero ciclo FDE viene completato in un clock.

le memorie realizzate con flip-flop sono dette SRAM, sono veloci ma costose per quanto riguarda il costo in bit.

SRAM vengono quindi usate per la cache , DRAM per la memoria centrale, e poi dischi.

il passaggio da una memoria all'altra si gestisce con paginazione.

Le DRAM o RAM dinamiche hanno un solo transister e un condensatore, che mantiene un bit.

visto che il condesatore puo perdere la propria carica, sono necessari periodi di refresh, cio le rende piu lente.

ci sono poi vari livelli di cache.

una piu piccola dentro al cip della CPU, una media nello stesso involucro della CPU e una piu grande esterna

con **localita temporale** si intende l'alta probabilita che la stessa cella venga acceduta in una breve distanza di tempo

con **localita spaziale** si intende l'alta probabilita che celle di memoria vicine vengano accedute a breve distanza di tempo.

esistono diversi metodi di gestione della cache. noi vediamo il "cache a corrispondenza diretta" o "direct mapped cache".

suddividi la memoria in blocchi di dimensione m.

crei n linee di cache di dimensione m. le linee sono iindicizzate da 0 a n-1.

blocchi di memoria vengono inseriti in certe linee di cache "ad orologio", e si tiene conto di quale blocco e dove.

ogni linea ha 1 booleano Valid che dice se la linea e occupta, data che contiene i dati e tag che indica il blocco della emoria principale contenuto.

quando l'accesso alla cache ha successo e il blocco di memoria e gia presente in cache, si dice che e avvenuto un cache hit, altrimenti e cache miss. se cache miss, il blocco deve essere portato in cache, qui ce un momento in cui dati di cache e dati di memoria centrale non corrispondono, cio puo creare problemi.

esistono gli stessi prooblemi tra memoria centrale e memoria di massa(dischi), qui si usa la paginazione.

a differenza del processore hack, istruzione e operandi spesso sono caricate insieme, questo richiede un ciclo di clock molto lungo.

spesso si pre-carica la prossimo istruzione mentre la prima e in esecuzione, questo e gestito dall' IFU(instruction fetch unit) Questo e un esempio di pipeline.

i salti pero possono essere problematici per le pipeline, visto che richiedono di scartare le istruzioni successive. quindi architetture moderne cercano di predire i salti.

ci sono anche problemi di concorrenza e accesso a sezioni critiche.

6 ISA del processore hack

per usare un processore lo si deve programmare tramite un suo linguaggio assembly, un set di istruzioni detto "instruction set".

il linguaggio assembly corrisponde ad istruzioni macchina subito eseguibili. e praticamente una versione leggibile del binario.

il linguaggio macchina supportato dall'architettura dipende dall'instrucion set architecture(ISA)

possiamo definire l'ISA come l'insieme degli elementi dell'architettura visibili al compilatore.

ci sono due approcci per migliorare le architetture: RISC e CISC

la hack non e facilmente classificabile.

hack e una macchina a 16 bit costituita da

- RAM: composta da registri a 16 bit. salva dati di lavoro
- ROM: composta da registri a 16 bit, contiene istruzioni dei programmi da eseguire.
- registro A
- registro D
- M: registro di memoria puntato da A
- ALU per eseguire poi le istruzioni.
- program counter: il registro che contiene l'indirizzo del registro ROM con la prossima istruzione.

esistono 2 tipi di istruzioni:

- A instruction: servono solo a caricare valori sul registro A.
- C instruction: istruzioni che eseguono una computazione prelevando dai registri A, D ed M e poi registrando il nuovo valore.

le istruzioni C hanno 3 parti:

- dest: specifica dove memorizzare la computazione.
- comp: specifica la computazione da eseguire.
- jump: specifica se, dopo avere eseguito dest=comp, bisogna saltare a un'altra istruzione. Lo fa dicendo di aggiornare il PC al valore del registro A.

cose da ricordare:

- M è sempre uguale a MEM[A], quindi se @3; M=MEM[3]
- mentre A e D sono registri della CPU, gli M sono celle di memoria della RAM.
- l'architettura non supporta A+M o M+A, devi sempre fare D=M; D=D+A.

- puoi fare A=M, ma attento che se fai un altro @ cancelli A

etichette: da usare come se fossero valori, usate per indicare destinazioni per i salti. divise in 2 parti: la dichiarazione, indicata da @, dove finisce il salto.

e l'uso di etichetta, da cui parte il salto, deve essere uguale al nome della destinazione.

7 componenti hack

chip memory composta da RAM(16k), chip screen(8k), usta per la mappa dello schermo. e chip keyboard, un singolo registro per il tasto usato.

lo schermo ha una funzionalita di tipo RAM,Esiste una corrispondenza diretta fra ogni pixel del monitor ed i corrispondenti bit memorizzati ad opportuni indirizzi all'interno del chip Screen.

il chip screen contiene 8k parole da 16 bit.

cpu contiene ALU e 3 registri, A,D, PC.

2 bit di controllo della ALU sono nr e zr

il primo bit del binario e 0 se si tratta di una A instruction.

7.1 assembler

assembler traduce assembly in linguaggio macchina(binario)

assembler hack traduce da ssembly hack a un file un numero binario a 16 bit per ogi riga in ASCII

symbol table: tabella con tutti i simboli, sia i predefiniti(R0....R15) che eventuali simboli, che vengono aggiunti man mano. a ognuno viene asseganto un valore.

processo di assemblaggio:

1. inizializzazione: si apre il file in input e si inizializza la symbol table con i simboli predefiniti.

2. prima passata: si passa l'input, inserendo etichette nella symbol table, e aggiungendo 1 a un contatore ogni volta che hai una A/C instruction. Quando incontri una definizione di etichetta(es: (loop)), le assegni il valore ddel contatore in quel momento +1 e non aumenti il contatore.

3. seconda passata: si apre in scrittura l'output e si assegna a ogni istruzione il suo valore. per le istruzioni con simbolo, si cerca in symbol table il valore del simbolo, se non ce(non e etichetta) gli si assegna un valore a partire da 16 e lo si memorizza in symbol table.

8 livello ISA

ora parliamo dell'ISA in modo piu generico.

livello ISA rappresenta l'interfaccia tra hardware e software, non comprende solo l'instruction set ma anche tutto cio che e visibile al compilatore ed e necessario per generare codice macchina.

formato di istruzioni: set di regole per facilitare l'implementazione logico-digitale, in hack erano A/C instruction.

in un computer generico, le istruzioni ISA contengono:

opcode o codice operatio, cioe l'istruzione da eseguire(somma,and ....)

e gli operandi, rappresentati attraverso i loro indirizzi.

a differenza di hack istruzioni diverse possono usare un numero diverso di bit per la codifica in binario, inoltre anche il nunero di operandi puo cambiare.

si puo segliere se usare piu bit per opcode o per gli indirizzi(operandi), determinando il numero di indirizzi o un numero maggiore di operazioni.

anche la dimensione delle intere istruzioni puo caambiare in base alla word. istruzioni piu piccole occupano meno memoria ma sono meno facili da decodificare.

se ad esempio una word è 32 bit, e divis in 4 byte, i byte poi sono salvati in celle di memoria di ordinamenti fissati.

l'ordine puo essere:

- big endian: 0-1-2-3
- little endian: 3-2-1-0

esistono diversi modi per indicare come reperire i vari operandi da usare:

- indirizzamento immediato: l'istruzione contiene gia l'operando da usare.
- indirizzamento diretto: l'istruzione contiene l'indirizzo completo della cella di memoria da cui prendere l'operando.
- indirizzamentoo a registro: l'operando viene preso da un registro.
- indirizzamento a registro indiretto: l'operando viene prelevato dalla memoria a un indirizzo puntato da un registro.
- indirizzamento indicizzato: l'istruzione contiene un "offset" che va sommata al contenuto di un registro per ottenere l'indirizzo di memoria.
- indirizzamento a stack: Si considera una parte di memoria da gestire secondo un modalità LIFO (last in-first out), ovvero tramite uno stack

inoltre ci sono 3 tipi di operazioni:

- trasferimento: è possibile trasferire dati da memoria a registri, da registri a memoria e da registro a registro.
- operazioni aritmetico-logico binarie: esistono anche operazioni con floating point oltre che interi.
- salti

procedura: serie di istruzioni che viene chiamata, esegue un compito, e torna al punto di partenza.

solitamente esistono istruzioni anche per la chiamata di procedura.

si ha la chiamata **CALL** che interrompe l'esecuzione sequenziale, tiene traccia del punto di partenza e comincia ad eseguire la procedura invocata.

il **return**(o ret) invece fa tornare a d eseguire l'istruzione dopo la chiamata, tipicamente inserendo l'indirizzo di ritorno nel program counter.

implementare le chiamate a procedure richiede inizializzare ed allocare la memoria necessaria ad eseguirle, tale spazio di memoria viene chiamato **activation record**(o record di attivazione) della procedura invocata.

i record di attivazione sono organizzati a stack(last in first out), quando termina una procedura si riattiva la precedente.

il record di attivazione si puo considerare l'insieme di tutti i dati necessari ad eseguire una singola chiamata di una procedura.

per gestire lo stack dei vari record di attivaazione, si usano 2 puntatori: FP(frame pointer) che punta all'inizio del record corrente e SP(stack pointer) che punta alla cima dello stack.

quando lo stack e vuoto SP=FP

poi man mano che aggiungi o togli funzioni o variabili SP cambia di continuo, FP e fisso ma cambia al cambio della funzione. in cima a ogni funzione, subito sotto a dove sarebbe SP e sopra all'indirizzo di ritorno, ce un puntatore al FP precedente che permette di aggiornare FP nel posto giusto al ritorno sul record precedente.

come conponenti del record di attivazione abbiamo:

- parametri
- indirizzo di ritorno: per rimettere in esecuzione il chiamante, cioe l'indirizzo dell'istruzione successiva al chiamante.
- vecchio frame pointer: Serve per ripristinare FP al momento del ritorno al chiamante
- variabili locali

questa organizzazione a stack permette di usare un'indirizzamento a stack e indicizzato per quanto riguarda al vslore corrente di FP.

serve uno stack per ogni thread in esecuzione.

ora esiste il problema di concorrenza, visto che ognuno di questi ha bisogno di uno spazio di memoria, ce il rischio che un programma scriva nella memoria sbagliata.

8.1 trap

trap: chiamata di procedura che si verifica al rilevamento di un errore rilevante.

fa trasferire il controllo a un processo detto **gestore di trap**

le trap sono quindi sincrone al programma, visto che vengono scatenate da un'istruzione del programma.

come le trap, gli **interrupt** sono condizioni che interrompono il programma per affidare il controllo a un gestore(detto interrupt service routine, o ISR).

gli interrupt pero non si riferiscono solo ad errori ma anche a notifiche ed eventi.

gli interrupt sono fondamentali per evitare il busy waiting.

gli interrupt sono invece asincroni, cioe sono scatenati da eventi indipendentemente dal programma in esecuzione.

Poi non e sempre bene che un interrupt interrompa l'esecuzione attuale, ad esempio l'ISR non dovrebbe essere interrotto da altri interrupt. Per questo il sistema operativo puo **mascherare** un interrupt, facendo in odo che non interrompa. Certi interrupt importanti possono essere dichiarati non mascherabili.

si puo anche assegnare una priorita ad ogni interrupt creando una gerarchia di priorita.

9 livello del sistema operativo

Le risorse messe a dispozisione dall'architettura vengono gestite dal sistema operativo(un software)

il sistema operativo e l'interfaccia tra il calcolatore e i programmi.

il sistema operativo gestisce tutto, dalla gestione della memoria, gestione dei processi, delle periferiche ecc....

abbiamo come step di interpretazione di un codice:

1. alto livello(livello 5). poi compilatore
2. assembly(A/C instruction)(livello 4). poi assembler.
3. poi livello macchina del SO(livello 3) poi SO.
4. ISA(come livello 4 ma binario)(livello 2) poi microprogrammazione
5. Livello microarchitettura(livello 1)(esecuziona da CPU)
6. Livello logico-digitale(seganli elettrici dentro porte logiche)(livello 0)

9.1 paginazione

negli anni 60 i computer avevano memoria RAM limitata e non riuscivano a usare appieno i loro bit, se ad esempio un computer ha 16 bit e 4k locazini, quindi ha solo 4096 indirizzi fisici rispetto ai 65535 che potrebbe avere.

gli indirizzi non fisici sono detti virtuali.

in ogni momento poi massimo 4k indirizzi sono posti in memoria primaria e il resto sono nella memoria secondaria.

dal punto di vista dell'accesso alla memoria, appare davvero da 64k locazioni, il problema e scambiare continuamente il contenuto tra le 2 memorie.

i pro principali della paginazione sono: possiamo avviare piu processi e questi hanno a disposizione piu memoria(puoi avere programmi piu grandi della ram ma ne puoi usare massimo l'equivalente della ram nello steso momento), ogni processo ha uno spazio di indirizzamento virtuale e cioli tiene separati,i programmi possono astrarsi dalle questioni implementative.

contro: la memoria secondaria e meno performante della memoria principale, creando un bottleneck, inoltre tutto cio richiede lavoro.

abbiamo 2 necessita: mappare gli indirizzi virtuali per poterli recuperare e gestire la sostituzione continua di pagine.

l'approccio per risolvere entrambi si chiama **paginazione**

sia le memoria principale che la secondaria sono divisi in parti di uguale dimensione. quelle dell primaria sono dette **blocchi**, quelle della secondaria sono dette **pagine**

se la dimensione di una pagina e da 2^k bit, k bit sono usati come dimensione dell'offset per trovare una locazione all'interno di una pagina.

bisogna **mappare**(cioe associare un indirizzo fisico a uno virtuale) gli indirizzi virtuali a cui si accede negli indirizzi fisici.

il mapping viene fatto dall' SO in modo trasparente al programma.

questo viene fatto dalla **memori management unit**(MMU) che, all'interno di una **tabella delle pagine** segna, per ogni pagina se questa e in memoria primaria e se si in quale blocco.

L'indice della pagina in cui cade l'indirizzo virtuale cercato si ottiene guardando l'indirizzo stesso, esclusi i k bit usati per identificare l'offset.

La tabella indica quali bit inserire al posto di questo indice di pagina.

l'indirizzo logico indicano gli indirizzi fisici. sono fatti cosi: indichiamo con 2^m lo spazio di indirizzamento totale(es 64k bit), mentre n indica sempre l'offset.

all'interno dei primi m bit dell'indirizzo logico si trova il numero della pagina, negli ultimi n bit si trova l'offset con cui trovare l'indirizzo.

9.1.1 page faults

il **page fault** è una trap che avviene quando si accede a una pagina non in memoria.

il gestore della trap ora deve scegliere quale pagina rimuovere per fare posto. questa sostituzione si dice **page swapping**

la scelta di quale algoritmo liberare viene presa da algoritmi di paginazione.

2 tipici algoritmi di paginazione:

- least recently used: si rimuove la pagina usata meno di recente, puo essere implementato come lista.
- FIFO: si rimuove la pagina in memoria principale da piu tempo, puo essere implementata tramite coda.

si dice virtual address space lo spaazio in memoria secondaria riservato a un processo. questo fa parte del suo " execution context".

context switch: l'interruzione e salvataggio di un processo per riprenderne un altro.

9.1.2 dirty bit

quando devi liberare un blocco, ma questo non e staato modificato, non e necessario ricopiarlo in memoria secondaria.

per tenere traccia di questo si usa la tecnica del **dirty bit**

quando si carica una nuova pagina in un blocco, si inizializza un certo bit a 0, se poi si accede in scrittura alla pagina nel blocco il bit diventa 1.

9.2 segmentazione

frammentazione interna: quando un programma richiede piu del multiplo preciso della dimensione delle pagine e rimangono dei byte liberi nell'ultima pagina.

puoi organizzare la memoria in segmenti,ciascuno con il proprio spazio di indirizzamento, in modo da usarne il piu possibile.

a differenza della paginazione, la frammentazione non e trasparente, il programma deve esserne a conoscenza e usarla.

ogni segmento di dimensione p usa gli indirizzi da 0 a p-1.

per distinguere segmenti di frammenti divers, gli indirizzi dovranno avere l'indicatore del relativo segmento.

registri di segmento, vengono usati per mantenere l'inizio di ogni segmento in memoria

i segmenti non usati tornano in memoria secondaria, ma al cambio con altri segmenti, possono riamanere altri bit vuoti, questi gruppi di bit vuoti si chiamano **lacune**

e il fenomeno della loro creazione e detto **frammentazzione esterna**, risolta tramite **compattamento**

ci sono diverse tecniche di cambio segmenti per cercare di ridurre la frammentazione esterna, 2 sono:

- best fit: si inserisce il nuovo segmento nella lacuna sufficente piu piccola
- first fit: si inserisce nella prima lacuna sufficente

per mitigare la frammentazione esterna, inoltre, si puo combinare paginazione e segmentazione. Quindi virtualizzare la memoria tramite paginazione, mantenendo in aggiunta la divisione logica offerta dai segmenti.

praticamente puoi dividere ogni segmento in pagine e poi riportare in memoria secondaria le pagine che non ti servono.

e pero molto lavoro per la MMU che deve tenere anche una tabella dei segmenti e una per ogni segmento paginato.

nella pratica poi la segmentazione varia molto in base ai casi. in fatti molti aspetti della paginazione che abbiamo visto sono superflui per i moderni pc a 64bit.

distinguiamo ad esempio 3 modalita di architettura

- modalita reale: i registri dei segmenti contengono l'indirizzo dell'inizio del segmento in memoria, e non c'è protezione degli accessi. Non c'è paginazione. Spesso usato ancora in avvio.
- Modalità protetta (CPU in protected mode, vedi architetture x86 a 32bit successive): i registri dei segmenti puntano a dei descrittori salvati in tabelle in memoria che, oltre all'indirizzo, definiscono modalità di accesso e dimensioni del segmento (max 4GB). Queste tabelle sono la Global Descriptor Table (GDT) per segmenti condivisi dall'intero sistema e le Local Descriptor Tables (LDT) per quelli locali ai processi. Può usare paginazione e richiede MMU.
- Long mode: nei sistemi a 64bit (vedi architetture x86-64 AMD e Intel) quando non in compatibility-mode, effettivamente si elimina la necessità dei segmenti vista la enorme capacità di indirizzamento. Si mantengono solo 2 segmenti generici, ma con indirizzo di inizio a 0 e senza dimensione massima. La protezione della memoria è garantita tramite paginazione (i processi hanno accesso solo alle proprie pagine, nelle quali si salvano le porzioni corrispondenti a code/heap/stack/etc), e la paginazione è obbligatoria e multilivello.

9.3 linking e collegamento dinamico

diverse parti di un programma possono essere compilate separatamente per creare moduli oggetto.

quindi poi serve una fase di collegamento fatta da un linker.

ogni modulo oggetto viene realizzato assumendo di usare il proprio spazio di indirizzamento a partire da 0,ma poi non sara piu cosi. bisogna quindi convertire questi indirizzi in indirizzi globali.

il linker decide l'ordine dei moduli, inserisce all'inizio informazioni di servizio per eseguire il programma, e poi prende nota della lunghezza di ogni modulo, e dell'inizio dell'indirizzo del nuovo spazio di indirizzamento.

e poi necessario cambiare gli indirizzi dentro i moduli oggetto, cio e detto **rilocazione**

per i riferimenti esterni servono piu informazioni.

le informazioni devono essere gia nell'assembly dei riferimenti,con cio e la tabella degli indirizzi di inzio, il linker puo rilocare anche le informazioni esterne.

In alcuni casi, le procedure che vengono usate, non vengono inserite nell'eseguibile: vengono collegate dinamicamente al momento dell'esecuzione del programma stesso.

si fa spesso con le librerie ad esempio.

un modo per farlo e generare volutamente una trap chiamando un indirizzo falso in una chiamata a una funzione dinamica. il programma fa la trap e cede al controllo al sistema operativo, questo legge il nome vero della funzione, la cerca , la trova e la patcha.

Un esempio e la DLL(dynamic link library) una libreria comune a moltissimi programmi, senza ognuno dovrebbe aver all'interno l'intera libreria.

10 virtual machine hack