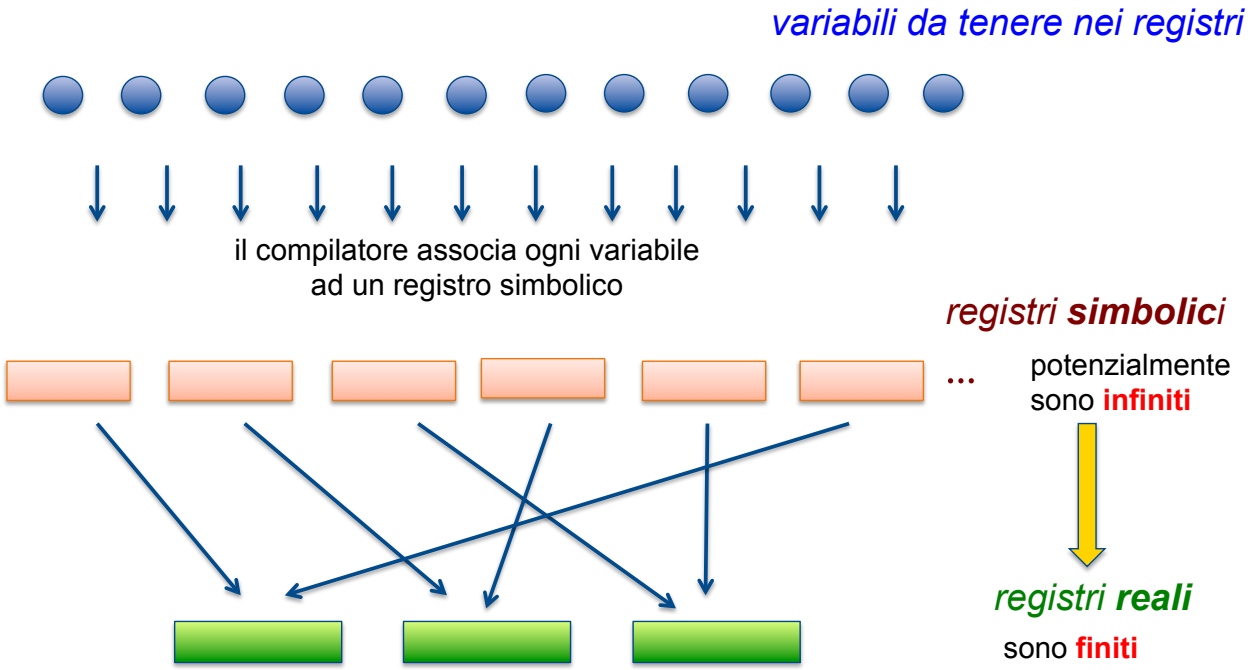
ottimizzazione dei registri

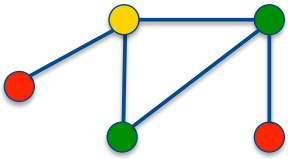
- l'architettura RISC può avere **pochi registri** (16-32) il cui uso viene **ottimizzato dal compilatore**
  - **scopo**: trovare gli operandi il più possibile nei registri e minimizzare le operazioni di load/store
  - Linguaggi ad alto livello non fanno riferimento esplicito ai registri, eccezione in C: `register int`



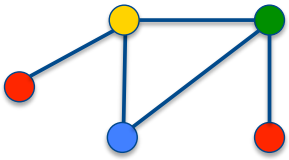
- il compilatore mappa ogni registro simbolico ad un registro reale
- se due registri simbolici **si usano in momenti diversi**, possono essere **mappati sullo stesso registro reale**
- se in un certo intervallo di tempo i **registri reali non sono in numero sufficiente** per contenere tutte le variabili riferite in quell'intervallo, alcune variabili vengono **mantenute nella memoria principale**

ottimizzazione dei registri

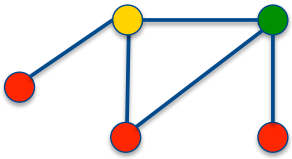
- decidere quale registro simbolico (quale variabile) assegnare a quale registro reale in ogni momento
- m compiti da eseguire, n risorse, con m >> n. Decidere quale compito assegnare a quale risorsa in ogni momento (es. m voli da effettuare, n aerei)
- equivale a risolvere un problema di **colorazione di un grafo:**
  - assegnare un colore ad ogni nodo in modo che
  - nodi adiacenti abbiano colori diversi
  - usare il minimo numero di colori



NO

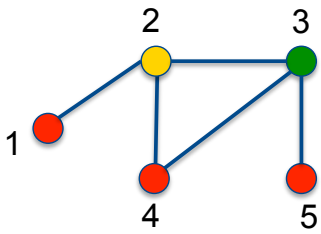


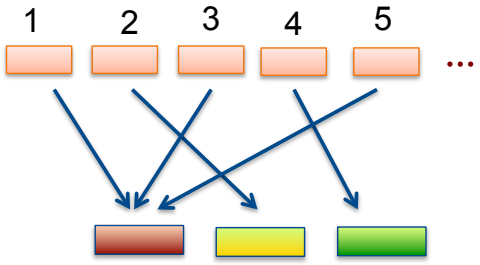
NO



SI

ottimizzazione dei registri

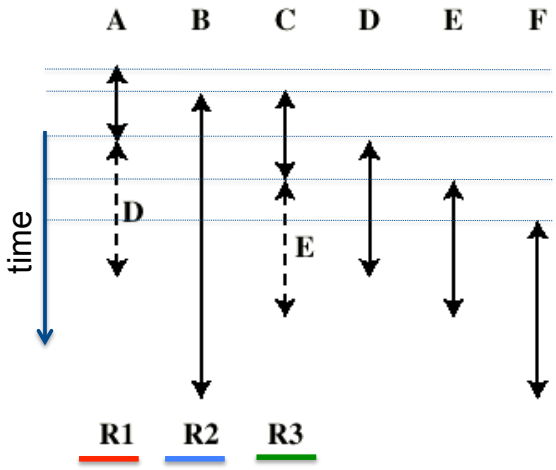




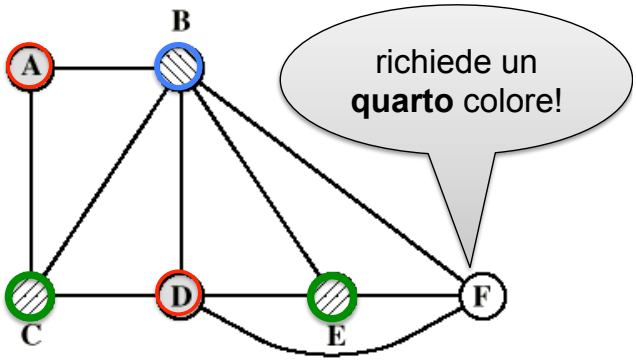
- i **nodi** corrispondono ai **registri simbolici**
- i **colori** corrispondono ai **registri reali**
- due nodi collegati da un **arco** se i due registri simbolici (variabili) sono **"in vita" nello stesso intervallo** di tempo/porzione di codice
- i nodi dello stesso colore possono essere assegnati allo stesso registro reale
- se servono più colori di quanti sono i registri reali, allora i nodi che non riescono ad essere colorati vanno memorizzati in memoria principale

grafo di interferenza

A,B,C,D,E,F registri simbolici e R1, R2, R3 registri reali



(a) Time sequence of active use of registers



(b) Register interference graph

colorazione del grafo di interferenza

- decidere se un grafo è colorabile con k colori è un problema “difficile” (NP-completo) nel caso generale.
- algorimi efficienti per casi specifici
- 32-64 registri fisici si dimostrano sufficienti

CISC o RISC?

- **CISC: ampio insieme di istruzioni, istruzioni più complesse**
  - per semplificare compilatore e migliorare *performance*
  - il compilatore deve generare "*buone" sequenze* di istruzioni macchina, cioè ***brevi*** e ***veloci*** da eseguire

CISC o RISC?

- **CISC: ampio insieme di istruzioni, istruzioni più complesse**
  - per semplificare compilatore e migliorare performance
  - il compilatore deve generare "*buone" sequenze* di istruzioni macchina, cioè brevi e veloci da eseguire
- **Sempifica il compilatore?**
  - istruzioni macchina complesse (quindi più simili a quelle HL) sono ***difficili da sfruttare***, perche' il compilatore deve trovare dei match precisi (è importante anche il **contesto i**n cui è inserita un'istruzione!)
  - con un set di istruzioni complesse è più ***difficile ottimizzare il codice macchina prodotto***, cioè **ridurlo** e **riorganizzarlo** per migliorare la pipeline

CISC o RISC?

- **CISC: ampio insieme di istruzioni, istruzioni più complesse**
  - per semplificare compilatore e r
  - il compilatore deve generare "*bu* china, cioè brevi e veloci da eseguire
- **Sempifica il compil**
  - istruzioni macchina uelle HL sono ***difficili da sfrutta*** vare dei match precisi (è importante anche il **contesto** in inserita un'istruzione!)
  - con un set di istruzioni complesse è più ***difficile ottimizzare il codice macchina prodotto***, cioè **ridurlo** e **riorganizzarlo** per migliorare la pipeline
  - le misurazioni dinamiche dicono che le istruzioni più frequenti sono le più semplici

si osserva un'istruzione di HL **nel suo contesto** (porzione di programma HL)

ottimizza **una sequenza di** istruzioni macchina

CISC o RISC?

- **CISC: ampio insieme di istruzioni, istruzioni più complesse**
  - per semplificare compilatore e migliorare performance
  - il compilatore deve generare "*buone" sequenze* di istruzioni macchina, cioè ***brevi*** e ***veloci*** da eseguire
- **Sequenze di istruzioni più brevi?**
  - sequenze brevi occupano meno memoria, ma la memoria è meno costosa
  - meno istruzioni implica meno fetch e più cache hit, quindi esecuzione più veloce
  - meno istruzioni **non significa meno bit di memoria occupata**:
    - molte istruzioni significa codici operativi più lunghi
    - riferimenti a registri richiedono meno bit dei riferimenti alla memoria
  - la **taglia** dei programmi compilati per RISC o CISC si dimostra **simile**

CISC o RISC?

- **CISC: ampio insieme di istruzioni, istruzioni più complesse**
  - per semplificare compilatore e migliorare performance
  - il compilatore deve generare "*buone" sequenze* di istruzioni macchina, cioè ***brevi*** e ***veloci*** da eseguire
- **Sequenze di istruzioni più veloci?**
  - un' istruzione complessa può essere eseguita più velocemente di una serie di istruzioni più semplici,
  - **ma**:
    - l'unità di controllo diventa più complessa
    - il controllo microprogrammato necessita di più spazio
    - quindi **si rallenta l'esecuzione delle istruzioni più semplici**, che restano le **più frequenti**

CISC o RISC?

caratteristiche di architetture **RISC**:

- **un' istruzione per ciclo di clock**
  - (*machine cycle*): tempo impiegato per fare fetch-decode-execute-write di un'istruzione elementare.
  - istruzioni RISC hanno un ciclo esecutivo che dura un solo ciclo di clock, quindi se la pipeline è piena, ad ogni ciclo di clock termina un'istruzione
  - istruzioni CISC richiedono più di un ciclo; un'istruzione RISC è veloce quanto le microistruzioni su macchine CISC
- **operazioni da registro a registro, tranne LOAD e STORE**
  - CISC attuali hanno anche operazioni *memory-to-memory* e *register/memory*
  - poiche' si usano di frequente scalari locali, *aumentando o ottimizzando i registri* la maggior parte degli operandi stanno a lungo nei registri.
- **pochi e semplici modi di indirizzamento**
  - indirizzo di registro, spiazzamento (relativo a PC)
  - si semplifica l'istruzione e l'unità di controllo

CISC o RISC?

caratteristiche di architetture **RISC**:

- **pochi e semplici formati fissi per le istruzioni**
  - campi e opcode a *dimensione fissa*, così la **decodifica dell'opcode** e **l'accesso ai registri** per gli operandi **possono essere simultanei**
  - istruzioni a lunghezza fissa sono **allineate con la lunghezza delle parole**, quindi il **fetch è ottimizzato** per prelevare (multipli di) una parola
  - la regolarità facilita le ottimizzazioni del compilatore
  - più responsivo agli interrupt, controllati tra due istruzioni più semplici
- **unità di controllo cablata**:
  - se cablata (cioè hardware) è meno flessibile ma più veloce
  - se microprogrammata più flessibile ma meno veloce

CISC o RISC?

- **non è evidente quale sia l'architettura nettamente migliore**
- **Problemi per fare un confronto:**
  - Non esistono architetture RISC e CISC che siano direttamente confrontabili
  - Non esiste un set completo di programmi di test
  - Difficoltà nel separare gli effetti dovuti all'hardware rispetto a quelli dovuti al compilatore
  - Molti confronti sono stati svolti su macchine prototipali e semplificate e non su macchine commerciali
  - Molte CPU commerciali utilizzano idee provenienti da entrambe le filosofie:
    - PowerPC architettura RISC con elementi CISC
    - Pentium II architettura CISC con elementi RISC

famiglia MIPS

- Studiamo i processori MIPS come esempio di architettura RISC
- architettura sperimentale sviluppata a Stanford negli anni ‘80 e poi sviluppata commercialmente
- Riferimenti: *Hennessy & Patterson “Struttura e progetto dei calcolatori”,* in Biblioteca

- Architettura molto regolare
- architettura progettata per un’implementazione efficiente della pipline
  - MIPS = ***m****icroprocessor without* ***i****nterlocked* ***p****ipeline* ***s****tages*

MIPS (a 32 bit)

Istruzioni:

- Tutte le istruzioni di dimensione 32 bit
- tutte le operazioni sui dati sono da registro a registro
  - le istruzioni che manipolano i dati si usano i valori dei registri
- le operazioni sulla memoria:
  - solo *load* e *store*, per trasferire dati tra memoria e registri
  - nessuna operazione memoria-memoria
- quindi tutte le istruzioni operano su registri, es `add $1, $2, $3`

Registri:

- 32 registri di 32 bit
- si indicano con $1, $2, $3…. **$0** contiene sempre 0