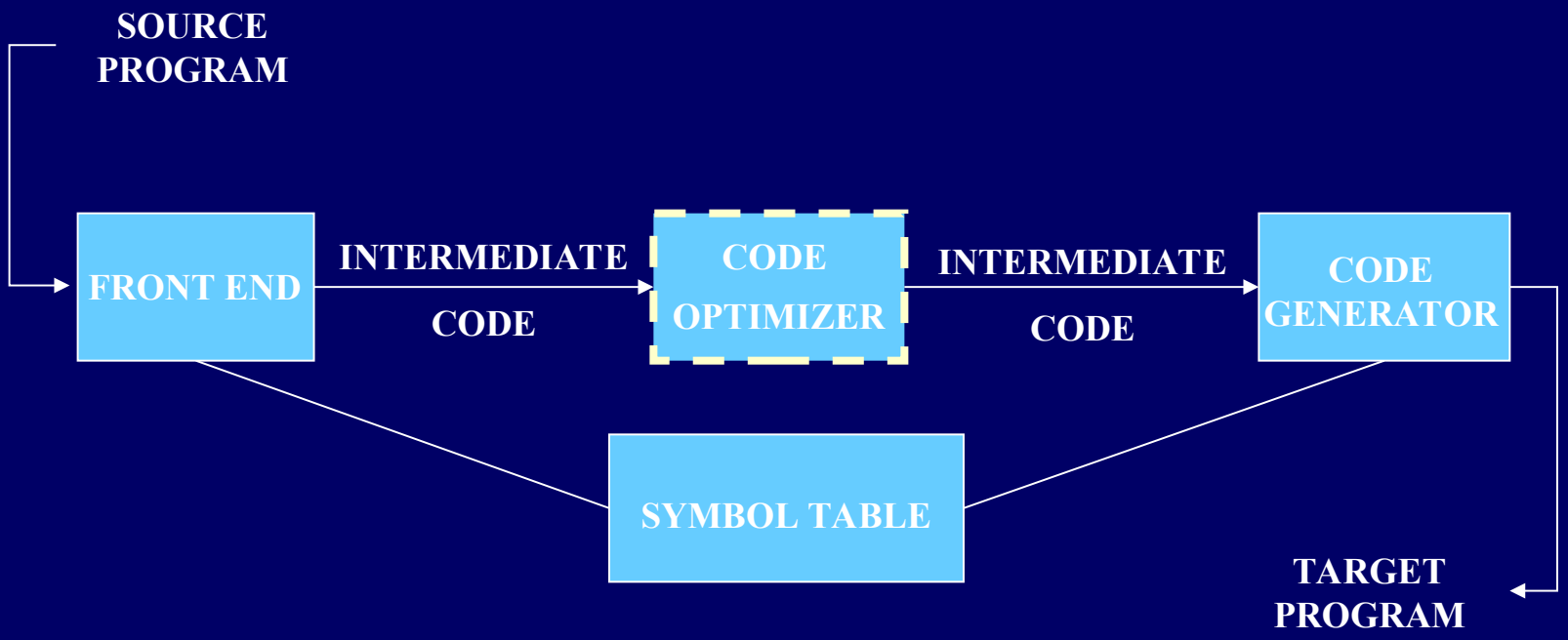
Generazione codice

Daniela Briola
Lorena Bellino

Struttura generale



Requisiti:

- Generare codice corretto
- Il generatore stesso deve essere efficente

Target program

- Linguaggio macchina assoluto
  - Subito posizionabile in una data locazione di memoria ed eseguibile
- Linguaggio macchina rilocabile
  - Necessario per compilazione a moduli
- Assembler

Allocazione della memoria

- Per le variabili, è basato sui dati inseriti nella symbol table nella fase precedente
- Per il codice, considerando istruzioni a tre indirizzi, si devono convertire le etichette in indirizzi di istruzioni
- Per ogni istruzione (quadrupla) si deve calcolare l'indirizzo della prima istruzione macchina associata (informazione che verrà poi mantenuta in un campo extra della quadrupla)
- Questo valore è dedotto semplicemente tenendo il conto delle parole utilizzate sino all'analisi di quell'istruzione

Allocazione della memoria: Esempio

- j: goto i
  - Tradotta banalmente se j>i
    - All'istruzione i è già stato assegnato un indirizzo, dunque l'istruzione j diventerà un semplice jump all'indirizzo macchina della prima istruzione di i
  - Più complessa altrimenti
    - L'istruzione i non è ancora stata tradotta → non si conosce il suo indirizzo macchina
    - Si mantiene, associata ad i, una lista in cui si inserisce l'indirizzo di j (e delle altre istruzioni che referenziano i)
    - Quando ad i sarà assegnato un indirizzo, si aggiorneranno anche le istruzioni interessate

Selezione delle istruzioni

- Dipende dall'uniformità e completezza del set di istruzioni macchina
- Per ogni tipo di istruzione a tre indirizzi si può creare un template per la trasformazione
- Per esempio, per l'istruzione x=y+z possiamo utlizzare
  - MOV y, R0 carica y nel registro R0
  - ADD z, R0 somma z ad R0
  - MOV R0, x salva in x il valore in R0

Selezione delle istruzioni (1)

- L'uso di questi template può però portare alla generazione di codice inutile o poco efficiente
- Esempio:
  - a=b+c, d=a+e

MOV b, R0	MOV a, R0
ADD c, R0	ADD c, R0
MOV R0, a	inutile
MOV a, R0	inutile, a è ancora in R0
ADD e, R0	ADD e R0
MOV R0, d	MOV R0, d

Selezione delle istruzioni (2)

- Se la macchina target offre molte istruzioni, è possibile che ci siano diversi modi di tradurre alcuni comandi. Alcune di queste traduzioni, pur rimanendo corrette, risultano magari essere molto inefficienti.
- Se per esempio la macchina target offre un'operazione di incremento, INC x, l'operazione a=a+1 potrebbe essere semplicemente tradotta con INC a, mentre per le regole viste prima sarebbe trasformata, inutilmente, in
  - MOV a, R0
  - ADD #1, R0
  - MOV R0, a

Allocazione dei registri (1)

- Per istruzioni che usano registri si hanno notevoli benefici:
  - Risultano essere più veloci
  - Occupano meno spazio di quelle relative ad operandi in memoria
- Dunque un uso accurato dei registri porta ad una conseguente efficienza del codice
- Purtroppo, la scelta ottimale dell'assegnazione registro-variabile è un problema NP-completo
- Inoltre, il SO e l'hardware impongono regole e restrizioni sull'uso di registri, stabilendo a priori una certa gestione di questi per alcune istruzioni

Assegnazione dei registri (2): sottoproblemi

- L'uso dei registri è solitamente suddiviso in due sottoproblemi:
  - Register allocation: si seleziona un sottoinsieme delle variabili che dovrà risiedere in un registro (non specificato) ad un certo punto del programma
  - Register assignment: si associa ad ognuna di queste variabili un certo registro

Assegnazione dei registri (3): esempi di restrizioni

- Alcune macchine richiedono una coppia di registri (di cui uno pari e il successivo, dispari) per l'esecuzione di un'istruzione
- Es: Moltiplicazione:
  - M x y
  - x è un registro pari
  - Il valore di x è preso dal registro dispari che è in coppia con il pari indicato da x
  - y è il moltiplicatore (un registro singolo)
  - Il risultato è memorizzato nella coppia di registri

Assegnazione dei registri (4): esempi di restrizioni

Esempio:

- t=a+b t=t*c, t=t/d

LOAD R1, a
ADD R1, b a questo punto t si riferisce a R1
M R0, c
(D R0, d)
STORE R1, t

- In R1 si carica a e gli si somma b
- A questo punto i registri utilizzati da M sono già implicitamente definiti: l'attuale valore di t è memorizzato nel registro dispari R1, dunque il primo parametro dovrà essere il rispettivo registro pari R0. Alla fine dell'operazione il risultato (il nuovo t) sarà mantenuto in R0 e R1
- D è la divisione, simile a M. Al termine il risultato si trova nel registro dispari

Scelta dell'ordine di valutazione

- L'ordine di esecuzione delle operazioni influisce sull'efficienza del codice generato (alcune sequenze richiedono un uso diverso dei registri…)
- La scelta dell'ordine ottimale è NP-completo
- Per semplicità valuteremo le istruzioni nell'ordine in cui ci sono state fornite dal generatore di codice intermedio

La macchina target

- La conoscenza della macchina target e delle sue istruzioni è necessaria per progettare un buon generatore di codice
- Tuttavia, non è possibile generare del codice di qualità capace di adattarsi ad ogni macchina possibile, senza entrare in una precisa analisi della macchina stessa
- Dunque, di seguito ci riferiremo ad una macchina con indirizzamento per byte, con parole da quattro byte e n registri e con istruzioni del tipo
  - Op source, destination
  - Op identifica l'operazione, gli altri due campi i dati

Metodi di indirizzamento

Modo	Forma	Indirizzo	Costo Agg.
Assoluto	M	M	1
Registro	R	R	0
Indicizzato		c(R)	c+cont(R) 1
Indiretto reg.		*R	cont(R) 0
Indiretto index		*c(R)	cont(c+cont(R)) 1

- c(R) indica un offset sommato all'indirizzo in R
- cont(a) indica il contenuto del registro o dell'indirizzo di memoria
- * ritorna il valore memorizzato nell' oggetto che lo segue

Metodi di indirizzamento: esempi

- MOV R0, M: salva il contenuto del registro R0 nella locazione di memoria M
- MOV 4(R0), M: salva nella locazione di memoria M il valore cont(4+cont(R0))
- MOV *4(R0), M: il valore memorizzato è cont(cont(4+cont(R0)))
- MOV #1, R0: carica la costante 1 nel registro R0

Costo istruzioni

- Il costo è il numero di parole dell'istruzione
- Assumiamo che il costo base sia 1
- Sommiamo 1 per quelle istruzioni con modalità indirizzamento per sorgente e destinazione
  - 0 se uso un registro
  - 1 se si usa memoria o costante (perché devo memorizzarli)
- Minimizzare la lunghezza di un'istruzione corrisponde a minimizzarne i tempi di esecuzione

“Basic blocks” e “Flow graphs”

- Basic block: è una sequenza di istruzioni successive eseguite sequenzialmente dalla prima all’ultima
- Un Flow graph è una rappresentazione del programma basata su un grafo diretto in cui i nodi rappresentano computazioni (BB) e gli archi il flusso di controllo

Nomi in BB

- Uno statement x:=y+z definisce x e usa y e z
- Un nome in un BB è “vivo” in un certo punto se verrà usato successivamente, nello stesso BB o in un altro

Algoritmo per suddivisione in BB

- Input: sequenza di istruzioni
- Output: lista di BB tali che ogni istruzione appartiene ad uno ed un solo BB

1. Comandi “leaders”:
   - Il primo comando
   - Ogni comando indirizzato da un goto (condizionato o no)
   - Ogni comando che segue un goto (condizionato o meno)
2. Per ogni leader si crea un BB contenente il leader e tutte le istruzioni successive sino al prossimo leader

Esempio di individuazione BB

```
Begin
    prod:= 0; i:= 1;
    do begin
    prod:= prod+a[i]*b[i];
    i:= i+1;
    end
    while i<20
end
```

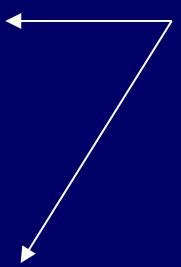
1) prod:=0
2) i:=1
3) t1:= 4*i
4) t2:= a[t1]
5) t3:= 4*i
6) t4:= b[t3]
7) t5:= t2 * t4
8) t6:= prod + t5
9) prod:= t6
10) t7:= i+1
11) i:= t7
12) if i <20 goto (3)

Trasformazioni su BB

- Trasformazioni che non trasformano la struttura:
  - Eliminazione di sottoespressioni comuni
  - Eliminazione di codice morto
  - Ridenominazione di variabili temporanee
  - Scambio di esecuzione di istruzioni adiacenti indipendenti
- Trasformazioni algebriche

Sottoespressioni comuni

a:= b+c
b:= a-d
c:= b+c
d:= a-d



Calcolano la stessa espressione

a:= b+c
b:= a-d
c:= b+c
d:= b

Notare che anche la prima e la terza istruzione sembrano uguali, ma la seconda ha modificato b, dunque le 2 istruzioni computano un valore differente

Eliminazione e ridenominazione

- Se in un BB si incontra un comando x:= u+j, e x risulti essere “morto” nella parte restante del BB, allora queste istruzione può essere eliminata
- Data un BB che definisce delle variabili temporanee, è possibile definire un BB equivalente dove si sia cambiato il nome ad ogni variabile temporanea con un nome nuovo.
  - Il BB di partenza è detto “normal-form” block

Scambio di istruzioni

Date le istruzioni

t1 := a+b

t2 := c*d

E' possibile invertirne l'esecuzione se e solo se "a" e "b" non sono "t1" e "c" e "d" non sono "t2"

Trasformazioni algebriche

- Le trasformazioni algebriche da un BB ad uno equivalente sono utili per semplificare espressioni o sostituire operazioni costose
- Es:
  - x:=x+0 o x:= x*1 possono essere eliminate
  - x:= y**2 (potenza) solitamente viene implementata con una chiamata di funzione. Può essere sostituita da x:= y*y

Flow graph

- E’ ottenuto aggiungendo informazioni sul flusso di esecuzione all’insieme dei BB:
  - I nodi del grafo sono i BB
  - Il nodo iniziale è quello del BB con la prima istruzione
  - Esiste un arco diretto da B1 a B2 se esiste un’esecuzione in cui B2 può seguire B1. Cioè:
    - Esiste un salto (condizionato o no) dall’ultima istruzione di B1 alla prima di B2
    - B2 segue immediatamente B1, e B1 non termina con un salto condizionato

Loop

- Un loop è un insieme di nodi del grafo tali che:
  - Formano una componente fortemente connessa
  - L'insieme ha un unico punto di ingresso: ogni nodo del loop non è raggiungibile dall'esterno se non passando per il nodo "entry"
- Un loop che non contiene al suo interno altri loop è detto "inner"

Rappresentazione dei BB

- Ogni BB può essere rappresentato da un record con:
  - Numero di quadruple (istruzioni) del blocco
  - Puntatore al leader
  - Lista dei blocchi predecessori
  - Lista dei blocchi successori
- Alternativamente, si può mantenere una lista likata delle quadruple
- Problema: il riferimento esplicito al numero della quadrupla a cui saltare nelle istruzioni di goto può creare problemi se la fase di ottimizzazione sposta le quadruple

Next-use information

- Si consideri che lo statement i assegni un valore a “x”
- Dato lo statement j che utilizza “x” come operando, se a partire da i non esiste un cammino sino a j in cui compaiano statement che modificano “x”, si dice che j usa il valore di “y” calcolato in i
- Si vuole calcolare, per ogni statement x:= y op z, quali saranno i prossimi usi di x, y e z

Next-use information (2)

- Supponiamo che anche le chiamate di procedura creino nuovi BB
- Scandendo in avanti le sequenza di statement si determina la fine del blocco
- Scandendo a ritroso si memorizza nella symbol table, per ogni nome, se ha un uso futuro o meno, e se è ancora vivo all'uscita del blocco
- Se la generazione del codice o l'ottimizzazione permette di usare certi temporanei in più blochi allora anche tali temporanei devono essere considerati vivi

Next-use information (esempio)

- Analizzando i: x:=y op z nella scansione a ritroso si eseguono le seguenti operazioni:
  - Si abbinano ad i le informazioni trovate nella symbol table relative all'uso futuro di x, y, z e il loro stato di esistenza (vive o meno)
  - Si imposta x a "not live" e "not-next-use"
  - Si impostano y e z a "live" e i "next-use" di y e z ad "i"

Allocazione di temporanei

- E’ utile nella fase di ottimizzazione creare un nuovo nome per ogni temporaneo
- Ogni temporaneo però necessita di spazio di allocazione
- Lo spazio complessivo per queste variabili tende a diventare troppo grande
- Dunque, si può ottimizzare posizionando due temporanei nella stessa locazione se non sono vivi contemporaneamente
- Si utilizzano le informazioni del ”next-use” per fare questa compressione

Allocazione di temporanei (2)

```
t1:= a*a
t2:= a*b
t3:=2*t2
t4:= t1+t3
t5:= b*b
t6:= t4+t5
```

```
t1:= a*a
t2:= a*b
t2:=2*t2
t1:= t1+t2
t2:= b*b
t1:= t1+t2
```

Un semplice generatore di codice

- Si suppone che ad ogni operatore nel codice a tre indirizzi corrisponda un'operazione nella macchina target
- Un valore (operando) è lasciato in un registro il più a lungo possibile, memorizzandoli in memoria solo
  - Se serve il registro per un'altra computazione
  - Prima di una call, salto o istruzione etichettata
- Si richiede di salvare in memoria ogni dato prima di uscire da un BB

Un semplice generatore di codice (esempio)

- Per esempio per l'istruzione a:= b+c possiamo generare:
  - ADD Rj, Ri lasciando a in Ri, se b e c erano nei registri e b morta (costo 1)
  - ADD c, Rj se c in memoria (costo 2)
  - MOV c, Rj utile se c verrà usato
    ADD Rj, Ri nel seguito (costo 3)

Descrittori di registri e locazioni

- Si usano per tener traccia dei contenuti di registri e locazioni:
  - Descrittore di registro: tiene traccia di cosa c'è all'interno di ogni registro. In ogni istante mantiene zero o più valori
  - Descrittore di locazione: tiene traccia della (delle) locazione in cui può essere ritrovato il valore associato ad un certo nome a run time. La locazione può essere un registro, una posizione nello stack, un indirizzo di memoria o un insieme dei tre. In generale è un'informazione associata alla symbol table

Algoritmo per la generazione di codice

Per ogni istruzione del tipo x:= y op z:

1. Invoca la funzione *getreg* per determinare la locazione L (registro o locazione di mem.) dove salvare il valore calcolato da y op z
2. Consulta il descrittore di y per determinare una locazione y' di y (possibilmente un registro se y ha tante locazioni). Se il valore di y non è già in L, genera MOV y', L

Algoritmo per la generazione di codice (2)

3. Genera l'istruzione OP z', L con z' locazione corrente di z.
   1. Aggiorna il descrittore di x indicando che la sua locazione è L
   2. se L è un registro, aggiorna il descrittore di L indicando che contiene il valore di x, e rimuove x da ogni altro descrittore di registro
4. Se i valori correnti di y e z non hanno next-use, non sono vivi all'uscita del blocco e sono memorizzati dentro registri, si devono aggiornare i relativi descrittori di registro per indicare che non contengono più i valori di y/z

Algoritmo per la generazione di codice (3)

- Dopo che si sono processate tutte le istruzioni, si devono memorizzare quei nomi che sono vivi all'uscita dal BB e non sono nelle loro locazioni
- Si usano i descrittori di registro per determinare quali nomi sono rimasti memorizzati nei registri, e si usano i descrittori di locazione per stabilire se queste variabili siano o meno già anche salvate in memoria (controllando che il nome sia vivo, altrimenti non deve essere salvato)
- Se nell'esecuzione non sono state computate informazioni di variabili vive, si deve assumere che tutti i nomi definiti dall'utente siano (ancora) vivi alla fine del blocco

Generazione codice

Daniela Briola
Lorena Bellino