Compilatore da pseudo-codice Pascal a Assembly

Lorenzo Dal Col

14 settembre 2006

Obiettivi

- Compilazione (traduzione) di programmi scritti in pseudo-codice Pascal in linguaggio Assembly
- Il codice generato deve essere perfettamente compilabile con MASM e LINK, ed ovviamente deve essere funzionante
- Lo pseudo-linguaggio deve supportare l'uso di variabili intere, char, e array monodimensionali di interi o char, inoltre ci deve essere un minimi supporto per le costanti stringhe, in modo da favorire la creazione di una minima interfaccia per i programmi da compilare
- Devono essere supportati i principali costrutti dei linguaggi ad alto livello: IF-ELSE, WHILE, LOOP, espressioni MATEMATICHE e BOOLEANE complesse, secondo la giusta priorità
- Si vogliono fornire al linguaggio anche procedure base native per l'IO con tastiera e display
- Gestione completa gli errori e visualizzare correttamente la riga e la tipologia dell'errore in modo accurato

Funzionamento (generico)

- Il compilatore riceve il nome del file scritto in pseudo-codice
- Le procedure che compongono lo SCANNER e il PARSER leggono un carattere alla volta il file e riconoscono parole, numeri, indici di array e parole chiave dello pseudo-linguaggio
- Ad ogni lettura completa di una “parola” essa viene memorizzato in una stringa VALUE e viene generato un TOKEN di una lettera che rappresenta la tipologia del simbolo stesso: x per le variabili, # per i numeri e i char, A per gli array, $ per le stringhe, e un’altra ventina di lettere per le parole chiave del linguaggio
- Una complessa sequenza di procedure correlate tra loro in un ciclo ricorsivo analizza il TOKEN, genera il codice Assembly opportuno, richiama lo scanner e il parser per leggere un’altra parola e il ciclo riparte
- Scrittura del file ASM oppure visualizzazione dell’ERRORE che è capitato durante la compilazione

Librerie utilizzate

- Per l’ IO da tastiera e su display sono state utilizzate le procedure standard della libreria “UTIL_PRG” e alcune procedure della libreria “GIOBE”
- Per l’analisi dei chars e il confronto e la copia delle stringhe sono state utilizzate delle procedure ricavate dalla “UCR Standard Library” promossa dal libro “The Art of Assembly Language”
- Per l’IO su file sono state create da zero delle procedure basate sull’INT 21h del DOS:
  fCreate, fOpen, fClose, fReadBuffer, fWriteBuffer, fGetC, fPutC
- I file vengono letti/scritti tramite un BUFFER di 256 byte in modo da velocizzare l’operazione
- Il BUFFER viene letto dallo SCANNER un carattere alla volta
- La gestione del BUFFER di IO dei files è automatica, si utilizzano solo fGetC e fPutC

Lo stack e i dati

- Lo STACK è impostato a un valore abbastanza grande, infatti:
- Durante la compilazione c'è la necessità di impilare molte stringhe, per gestire correttamente la ricorsione delle procedure di analisi dei blocchi di codice.
- Per impilare nello stack le stringhe sono state create due procedure, PushString e PopString:
  -la prima fa il PUSH di tutti i caratteri della stringa (SI) al rovesco, e mette in alto le due word necessarie all'istruzione RET (CS,IP)
  -PopString fa il POP di ogni carattere salvandoli in SI, finchè non viene trovato il carattere 0
- Il segmento dei DATI contiene le variabili utilizzate dal compilatore, la tabella dei simboli, la lista delle parole chiave dello pseudo-linguaggio, tutte le stringhe di errore e tutte le stringhe di codice Assembly utilizzate per la generazione del codice.

Lo Pseudo-Linguaggio Pascal (1)

- Per capire a fondo il meccanismo di compilazione è stata studiata a fondo la struttura di un linguaggio ad alto livello attraverso l'uso delle grammatiche BNF
- Ad esempio per quanto riguarda l'assegnazione di una variabile è stata ricavata la seguente grammatica:
- *<not_factor> ::= [!] <factor>*
  <factor> ::= <variable> | <constant> | '(' <expression> ')'
  <signed_term> ::= [<addop>] <term>
  *<term> ::= <not_factor> (<mulop> <not_factor>)**
  *<expression> ::= <signed_term> (<addop> <term>)**
  <assignment> ::= <variable> '=' <expression>
- La quale riesce a teorizzare bene un'assegnazione del tipo:
- **x=3+(5/a)-((3/4+1)*x)+x**
- Le procedure del compilatore sono scritte in modo da rispecchiare in modo diretto ed ottimale le teorie BNF, in questo modo è stato possibile arrivare a un alto grado di astrazione senza incorrere in troppi errori di programmazione

Lo Pseudo-Linguaggio Pascal (2)

- Per semplificare la compilazione è stato deciso di associare sempre ad ogni costrutto una parola chiave di chiusura:
  - If-else-ENDIF
  - While-ENDWHILE
  - Loop-ENDLOOP
  - Program-END
- Il linguaggio inoltre è stato semplificato i più possibile:
- I caratteri ; alla fine delle istruzioni sono FACOLTATIVI, servono esclusivamente a delimitare i commenti monoriga
- Il simbolo = è usato sia per l'assegnazione che per la relazione
- Il linguaggio NON è case-sensitive
- Le variabili INT e CHAR sono identiche, mentre le funzioni di lettura e scrittura variano a seconda se si vuole leggere un intero o un carattere.
- Le stringhe vengono dichiarate all'inizio e sono costanti, possono essere solamente visualizzate con un apposito comando

Limiti

- A causa della mancanza di tempo per proseguire con il progetto il compilatore attuale ha i seguenti limiti che possono essere facilmente superati:
- Mancano variabili e operazioni in Floating Point (sarebbe sufficiente doppiare tutte le procedure attualmente presenti per gli INT e convertirle)
- Gli array sono solamente monodimensionali (sarebbe sufficiente arricchire il parsing dell'indice [ ][ ] e calcolare la posizione fisica in memoria)
- Manca un supporto alle librerie standard già pronte
- **Il limite più grave e difficile da superare è il fatto che non esistono sotto-procedure e sotto-funzioni.**
- Supportarle presupporrebbe vari cambiamenti al "core" del compilatore, diventa necessario inserire il concetto di variabili globali/locali, il passaggio dei parametri per valore e riferimento, e quindi anche la generazione di codice per l'allocazione dinamica delle variabili, per poter fare anche funzioni ricorsive

Risultati

- Sono stati raggiunti ottimi risultati, con un supporto completo per:
- Espressioni matematiche e booleane anche complesse, fra immediati e variabili, garantendo la priorità degli operatori e le parentesi
- Supporto completo per gli array monodimensionali, compreso la compilazione degli indici calcolati array[<espressione>]
- Compilazione dei costrutti classici anche notevolmente annidiati: IF-ELSE, WHILE, LOOP, READ,READINT,WRITE,WRITEINT
- Supporto base per le stringhe
- Gestione degli errori avanzata: viene segnalata la riga, la tipologia di errore e spesso anche la parola chiave aspettata
- Il codice generato è Macro Assembler perfettamente compilabile
- Il compilatore è scritto in modo altamente modulare, rendendo possibile il futuro supporto per nuovi costrutti senza dover intervenire nel funzionamento centrale

Programmi compilabili

- Si possono compilare tutti i classici algoritmi usati per l'apprendimento della programmazione:
  - Alg. di ordinamento
  - Alg. di ricerca operativa a programmazione lineare
  - Alg. Greedy (Es. Knapsack problem)
  - MCD,mcm,numeri primi (è sufficiente implementare un ulteriore operatore RESTO)
- Grazie al costrutto MASM-ENDMASM è possibile anche inserire nel mezzo dello pseudo-codice anche del codice Assembly in modo di accedere anche a tutte le funzionalità del calcolatore non previste dallo pseudo-linguaggio.

Bibliografia

- The Art of Assembly Language - Randy Hyde
- Compiler Building Tutorial - Jack W. Crenshaw
- webster.cs.**ucr**.edu (UCR Standard Library)
- Compilers, Principles, Techniques, and Tools by Aho, Sethi, and Ullman