Laborator 3

Lucrarea prezintă limbajul de asamblare MIPS, procesul de transformare al unui cod în limbaj de asamblare în cod executabil de către microprocesor, precum și câteva exemple de simboluri folosite.

3.1 Limbajul de asamblare

Codarea instrucțiunilor sub forma de numere binare este naturală și eficientă pentru calculatoare. Oamenii în schimb au dificultăți în a interpreta aceste numere binare. Oamenii citesc și scriu simboluri mult mai ușor decât secvențe de numere binare.

Limbajul de asamblare este reprezentarea simbolică a codificării binare a unui computer, limbajul mașină. Limbajul de asamblare este mai lizibil decât limbajul mașină deoarece folosește simboluri în loc de biți. Simbolurile din limbajul de asamblare denumesc modele de biți care apar în mod obișnuit, cum ar fi codurile opționale sau regiștrii operand, astfel încât oamenii să le poată citi și memora. În plus, limbajul de asamblare permite programatorilor să utilizeze etichete pentru a identifica și numi anumite cuvinte de memorie care conțin instrucțiuni sau date.

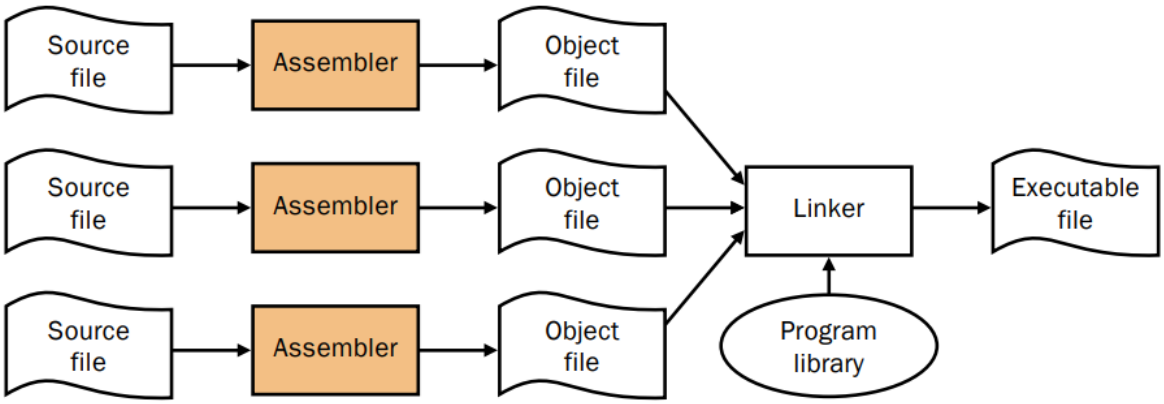

Figura 1 Procesul de creare al unui fișier executabil. Un program asamblor transformă un fișier scris în limbaj de asamblare într-un fișier obiect, care mai departe este conectat împreună cu alte fișiere obiect și fișiere bibliotecă într-un fișier executabil.

Un program numit **asamblor** traduce codul scris limbaj de asamblare în instrucțiuni binare. Acest tip de program oferă posibilitatea de reprezentare mai prietenoasă a instrucțiunilor pentru un microprocesor decât șirurile de 0 și 1, reprezentare care simplifică scrierea și citirea programelor. Numele simbolice pentru operații și locații de memorie sunt o parte a acestei reprezentări. O altă parte sunt facilitățile de programare care sporesc funcționalitatea unui program. De exemplu se permite unui programator să extindă limbajul de asamblare prin definirea de noi operații utilizând macrouri.

Un **fișier obiect** este un fișier ce conține codul obiect, adică codul mașină generat de un asamblor sau de un compilator. Codul obiectului este de obicei relocabil și, de obicei, nu poate fi executat direct. Există

diferite formate pentru fișierele obiect și același cod mașină poate fi ambalat în diferite formate de fișiere obiect.

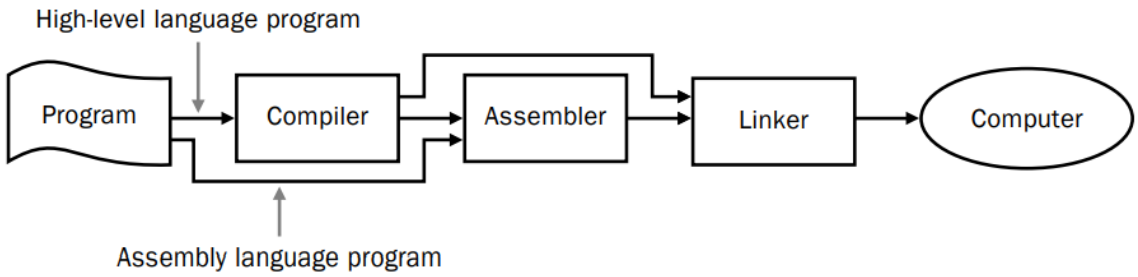

Figura 2 Codul în limbaj de asamblare este fie scris de un programator, fie este rezultatul unui compilator.

3.2 Simboluri folosite în limbajul de asamblare MIPS

```
        .text
        .align 2
        .globl main
main:
        subu $sp, $sp, 32
        sw $ra, 20($sp)
        sd $a0, 32($sp)
        sw $0, 24($sp)
        sw $0, 28($sp)
loop:
        lw $t6, 28($sp)
        mul $t7, $t6, $t6
        lw $t8, 24($sp)
        addu $t9, $t8, $t7
        sw $t9, 24($sp)
        addu $t0, $t6, 1
        sw $t0, 28($sp)
        ble $t0, 100, loop
        la $a0, str
        lw $a1, 24($sp)
        jal printf
        move $v0, $0
        lw $ra, 20($sp)
        addu $sp, $sp, 32
        j $ra

        .data
        .align 0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

3.3 Directive de asamblare MIPS

Directivele de asamblare sunt acele simboluri din limbaj ce îi indică asamblorului cum să transforme codul în continuare. Directivele de asamblare nu produc instrucțiuni și sunt precedate de caracterul ‘.’

Tabel 1 Directive MIPS ce sunt suportate de SPIM.

Sintaxă	Descriere
.ascii str	Stochează stringul *str* în memorie
.asciiz str	Stochează stringul *str* în memorie și adaugă stringul null la sfârșit.
.byte b1,..., bn	Stochează *n* valori *b1..bn*, în *n* bytes consecutivi de memorie.
.data <addr>	Elementele următoare acestei directive vor fi stocate sau vor face referire la segmentul de date din memorie. Dacă argumentul *<addr>* este prezent, atunci elementele vor fi stocate începând cu adresa *addr*.
.double d1,...,dn	Stochează *n* valori în virgulă mobilă cu dublă precizie în locații succesive de memorie.
.float f1,..., fn	Stochează *n* valori în virgulă mobilă cu simplă precizie în locații succesive de memorie.
.globl sym	Declară eticheta *sym* ca fiind globală, putând fi accesată și din alte fișiere
.half h1, ..., hn	Stochează *n* valori *w1..wn*, în *n* halfwords consecutivi(1halfword=2bytes)
.space n	Alocă n bytes în segmentul de date începând cu locația curentă.
.text <addr>	Elementele următoare vor fi adăugate în segmentul de text (unde sunt stocate instrucțiunile). Elemente pot să fie doar instrucțiuni sau directive *.words*. Dacă argumentul *<addr>* este prezent, atunci elementele vor fi stocate începând cu adresa *addr*.
.word w1,..., wn	Stochează *n* valori *w1..wn*, în *n* words consecutivi(1word=4bytes=32bits)

Exerciții și întrebări:

1. Care este diferența între directivele *.ascii* și *.asciiz*? Creați un fișier în care să folosiți cele două directive și rulați codul în SPIM.
2. Stocați numerele 2, 255, 400, 18500, 65534, 120000 în segmentul de date. Utilizați directiva potrivită în fiecare caz.
3. Comparați stocarea unor nume reale prin folosirea *.double* si *.float*