תקציר הפרוייקט – אסמבלר

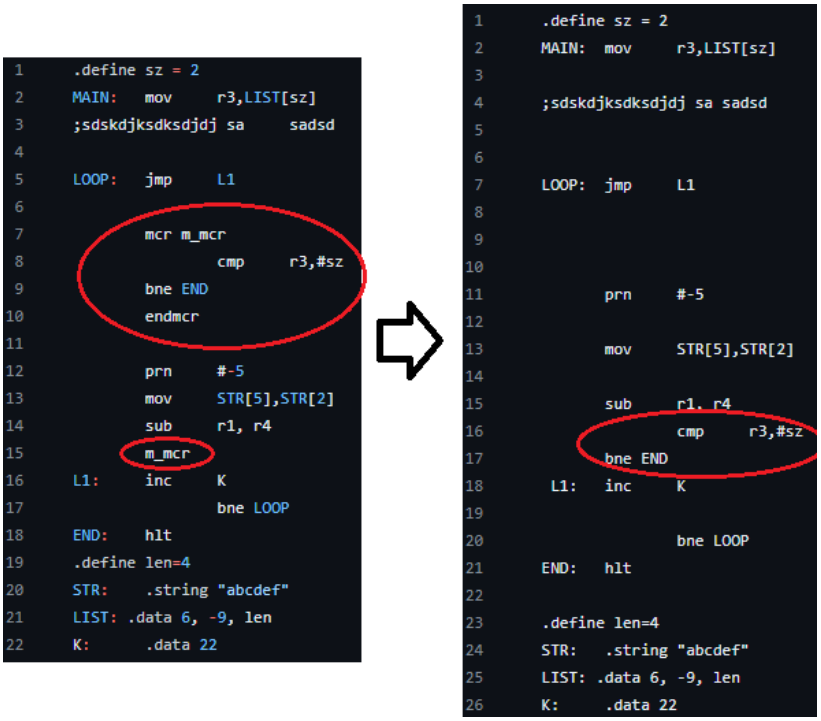
אסמבלר לשפת תכנות המוגדרת בפרויקט הגמר של קורס מעבדה בתכנות מערכות בשפת C של האוניברסיטה הפתוחה (ישראל).

בנוסף לתקציר זה קיים גם הסבר מפורט יותר לפרוייקט בגיט (כולל תמונות וקבצים נוספים)..

תפקוד התוכנה ושלביה:

שלב המאקרו:

האסמבלר מנתח את קובץ קוד המקור המקורי (.as) ומייצר את קובץ המקור המורחב (.am) המכיל את אותו תוכן קוד מקור, ההבדל היחיד הוא שפקודות המאקרו המוגדרות בקובץ המקור המקורי (.as)מוחלפות בתוכן שלהם, כלומר המאקרו נפרש. לשם כך, האסמבלר משתמש בטבלה כדי לאחסן את שם המאקרו עם האינדקסים ההתחלתיים והסיום שלו בקובץ. אם קיימת שגיאה כלשהי בתחביר של פקודות המאקרו האסמבלר יוציא הודעת שגיאה רלוונטית ויקפוץ ישר לקובץ הבא, אחרת, הוא ממשיך לשלב המעבר הראשון.



מעבר ראשון:

בשלב זה, האסמבלר מתחיל להרכיב את הקוד הבינארי, לשם זה הוא סופר את גודל תמונת הנתונים ותמונת הפקודות, מוסיף את כל הסמלים לטבלת הסמלים, ואחרון חביב האסמבלר מוודא שאין שגיאות בקוד. אם האסמבלר נתקל בשגיאה כלשהי בקוד הוא מוציא הודעת שגיאה/אזהרה ל-stderr כדי לדווח על שגיאה כלשהי בקוד המקור במעבר הראשון, האסמבלר מנתח את כל התוכן של קוד המקור ללא קשר לכל התרחשות של שגיאה על מנת למצוא שגיאות נוספות. אם מתרחשת שגיאה כלשהי, האסמבלר לא ימשיך לשלב הבא אלא ימשיך ישירות לקובץ הבא (אם קיים).

בשלב זה אם האסמבלר נתקל בתווית לא מוכרת (שאולי תוגדר בהמשך הקובץ) חלק זה בקוד יסומן כלא ידוע עד המעבר השני שם ישלים האסמבלר את חלק זה בקוד הבינארי.

Decimal Address	Source Code	Explanation	Binary Machine Code
0100 0101 0102 0103	MAIN: mov r3, LIST[sz]	First word of instruction Source register 3 Address of label LIST (integer array) Value of sz (index 2)	00000000111000 00000001100000 ? 00000000001000
0104 0105	LOOP: jmp L1	 Address of label L1	00001001000100 ?
0106 0107	prn #-5	 Immediate value -5	00001100000000 11111111101100
0108 0109 0110 0111 0112	mov STR[5], STR[2]	 Address of label STR (string) Index 5 Address of label STR Index 2	00000000101000 ? 00000000010100 ? 00000000001000
0113 0114	sub r1, r4	 Source register 1 and target register 4	00000011111100 00000000110000
0115 0116 0117	cmp r3, #sz	 Source register 3 Value of sz (immediate #2)	00000001110000 00000001100000 00000000001000
0118 0119	bne END	 Address of label END	00001010000100 ?

מעבר שני:

במעבר השני אנו משלימים את הקוד הבינארי, לשם כך האסמבלר כותב את המילים של כל שורת קוד בפורמט בינארי, מכניס את המילים (סיביות בינאריות) לתמונת הזיכרון במיקום הנכון בהתאם לסוג המילה (נתונים/הוראה) ומוסיף כל אופרנד חיצוני. שהופיע בקובץ. אם האסמבלר נתקל באופרנד תווית שאינו בתוך טבלת הסמלים ואינו חיצוני הוא מוציא הודעת שגיאה ולאחר מכן ממשיך לבדוק את שאר הקוד כדי לגלות את כל השגיאות מהסוג הזה ולדווח עליהן ובמקרה זה, יסיים את המעבר השני ולא יווצרו קבצים.

בסוף המעבר השני מתבצעת גם הצפנה של הקוד הבינארי.

Decimal Address	Source Code	Binary Machine Code
0115 0116 0117	cmp r3, #sz	00000001110000 00000001100000 00000000001000
0118 0119	bne END	00001010000100 00000111110010
0120 0121	L1: inc K	00000111000100 00001000011110
0122 0123	bne LOOP	00001010000100 00000110100010
0124	END: hlt	00001111000000
0125	STR: .string "abcdef"	00000001100001
0126		00000001100010
0127		00000001100011
0128		00000001100100
0129		00000001100101
0130		00000001100110
0131		00000000000000
0132 0133 0134	LIST: .data 6, -9, len	00000000000110 11111111110111 00000000000100
0135	K: .data 22	00000000010110

יצירת הקבצים:

אם המעבר השני הסתיים ללא כל שגיאה ניצור את כל הקבצים הנדרשים (קובצי ..ob, .ext .ent)

מטרות כלליות בפרוייקט:

יעילות זיכרון: היה לי חשוב להקצות את הזיכרון של פלט התמונה הבינארית בצורה דינמית כך שהשימוש בזיכרון בתוכנה יהיה יעיל ושהקצאת הזיכרון של התמונה שהאסמבלר יוצר תתאים למינימום הנדרש להידור קוד מקור.

הפרדת קבצים: פיצול התוכנית לרכיבים בלתי תלויים שיהיו אחראים למשימות מוגדרות היטב וששיתוף הנתונים בין קבצים יהיה באמצעות מבני נתונים.

הפשטה של מבני נתונים ומשתנים: כדי להתאים לצרכים שלנו על ידי הגדרת מבנים שונים מכמה סוגים.

בנוסף לתקציר זה קיים גם הסבר מפורט יותר לפרוייקט בגיט (כולל תמונות וקבצים נוספים)..

קישור לגיט

(כרגע עלה רק חלק קטן מהפרוייקט המדגים את עבודת האסמבלר – בהמשך לאחר אישור האוניברסיטה אפרסם את הפרוייקט המלא.)