**דוקומנטציה – מבנה המחשב – פרויקט 1**

**מיכאל מיאסקובסקי**

**דין כרמון**

**תוכן עניינים:**

אסמבלר- עמ '2

סימולטור- עמ '11

לוח הכפל- עמ' 19

22 'עמ -Bubblesort

24 'עמ -Mergesort

# אסמבלר

מטרת התוכנית: המרת קוד אסמבלי לקוד הקסדצימלי )הוראות מפורטות בקובץ מטלת הפרויקט(.

**תחילת הקוד** נתחיל מתחילת הקוד:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <math.h>

#include <stdbool.h>

כאן אנו מייבאים ספריות שיידרשו למימוש האסמבלר, למשל ספריית ה-stdio שבעזרתה ניגש לקבצים ,ספריית ה-string שדרכה נטפל במחרוזות וכן הלאה.

#define memorySize 65536

#define outputLineLength 8

#define maxLineLength 500 #define maxLabelLength 50

כמו כן הגדרנו מספר ערכים קבועים הנדרשים למימוש:

memorySize הוא מספר השורות בזיכרון, outputLineLength זה אורך שורה בזיכרון , maxLineLength זהו האורך המקסימלי של שורה בקובץ הקלט,

maxLabelLength זהו האורך המקסימלי של label. **פעולות** • הפעולה הראשונה בקוד היא פעולת ה-RemoveComment:

void RemoveComment(char \*s)

{

//Removes all comments in the assembly code. for (unsigned int i = 0; i < strlen(s); i++)

if (s[i] == '#')

{

s[i] = '\0'; break;

}

}

בפעולה זו אנו מוחקים הערות מקוד האסמבלי, על מנת שלא יפריעו להמרתו לקוד הקסדצימלי. עושים זאת על ידי הפיכת תו הסולמית לתו של סיום מחרוזת, מפני שסולמית מסמנת תחילת הערה. הקלט של הפעולה הוא המחרוזת, ובמקום לפלוט מחרוזת היא מבצעת את הפעולה ישירות על הפלט, כך שלאחר ריצת הפונקציה המחרוזת המקורית הופכת למחרוזת לאחר הורדת ההערה.

• הפעולה הבאה הינה פעולת ה-DecimalToBinary:

void DecimalToBinary(char \*s, char \*converted, int len)

{

//Converts a decimal string to a binary string.

int data = atoi(s);

for (unsigned int i = 0; i < len; i++)

{

unsigned int mask = 1u << (len - 1 - i); converted[i] = (data & mask) ? '1' : '0';

}

converted[len] = '\0';

}

פעולה זו מקבלת כקלט את המחרוזת s שהיא קוד דצימלי, מחרוזת converted ריקה ומספר len המייצג את אורך הקידוד הבינארי שנכניס למחרוזת converted. הפעולה עורכת את converted כך שיכלול את המרת הקוד הדצימלי ב-s לבינארי. בדומה לפעולה הקודמת, אין פה החזרת פלט, אלא עבודה על מחרוזות הקלט.

• הפעולה הבאה בקוד היא BinToHex:

void BinToHex(char \*binCode, char \*hex)

{

//Converts binary code to hexadecimal.

char part[5]; for (unsigned int i = 0; i < strlen(binCode); i += 4)

{

memcpy(part, &binCode[i], 4);

part[4] = '\0'; sprintf\_s(hex + i / 4, strlen(binCode) / 4 - i / 4 + 1, "%X",

(int)strtol(part, NULL, 2));

}

hex[strlen(binCode) / 4] = '\0';

}

פעולה זו ממירה קוד בינארי להקסדצימלי, על ידי שימוש בפעולה strtol. משתמשים ב-part לשם שמירה של 4 תוים בקוד הבינארי )השקולים לתו אחד בקוד הקסדצימלי(, ולאחר מכן ממירים אותו לבינארי ומחברים למחרוזת hex הכוללת את הקוד ההקסדצימלי עד לאותו חלק.

• הפעולה הבאה היא פעולת ImmToHex הממירה את ערך ה-imm בקוד האסמבלי לקוד הקסדצימלי של 4 תווים:

char \*ImmToHex(char \*s)

{

//Using the conversion functions, it returns a conversion of the imm code to

hexadecimal.

if (s[1] == 'x'&&s[0] == '0')

return s + 2;

else {

//If the imm is decimal it is converted using the decimal converter.

char converted[17];

DecimalToBinary(s, converted, 16); BinToHex(converted, s);

return s;

}

}

מתבצעת בדיקה אם קוד הקלט s הוא דצימלי או הקסדצימלי )על ידי בדיקת שני התווים הראשונים(. במקרה שהקוד הקסדצימלי מחזירים אותו ללא התוים 0x, ובמקרה שהוא דצימלי מבצעים המרה לקוד בינארי, ולאחר מכן להקסדצימלי על ידי הפעולות דלעיל והמחרוזת מוחזרת.

• הפעולה הבאה היא פעולת TextToHex הממירה מילים שמורות לקוד הקסדצימלי:

char \*TextToHex(char \*s)

{

//Converts the known words to binary.

if (strcmp(s, "add") == 0 || strcmp(s, "$zero") == 0)

return "0";

if (strcmp(s, "sub") == 0 || strcmp(s, "$at") == 0)

return "1";

if (strcmp(s, "and") == 0 || strcmp(s, "$v0") == 0)

return "2";

if (strcmp(s, "or") == 0 || strcmp(s, "$a0") == 0)

return "3";

if (strcmp(s, "sll") == 0 || strcmp(s, "$a1") == 0)

return "4";

if (strcmp(s, "sra") == 0 || strcmp(s, "$t0") == 0)

return "5";

if (strcmp(s, "limm") == 0 || strcmp(s, "$t1") == 0)

return "6";

if (strcmp(s, "beq") == 0 || strcmp(s, "$t2") == 0)

return "7";

if (strcmp(s, "bgt") == 0 || strcmp(s, "$t3") == 0)

return "8";

if (strcmp(s, "ble") == 0 || strcmp(s, "$s0") == 0)

return "9";

if (strcmp(s, "bne") == 0 || strcmp(s, "$s1") == 0)

return "A";

if (strcmp(s, "jal") == 0 || strcmp(s, "$s2") == 0)

return "B";

if (strcmp(s, "lw") == 0 || strcmp(s, "$gp") == 0)

return "C";

if (strcmp(s, "sw") == 0 || strcmp(s, "$sp") == 0)

return "D";

if (strcmp(s, "jr") == 0 || strcmp(s, "$fp") == 0)

return "E";

if (strcmp(s, "halt") == 0 || strcmp(s, "$ra") == 0)

return "F";

//In case the input is not a known word, the function returns the input. return s;

}

הפונקציה בודקת אם הקלט s נכלל במילים השמורות, ומחזיר תו הקסדצימלי לפי ערכו. אם הקלט לא נכלל במילים השמורות הפונקציה מחזירה את הקלט ללא שינוי.

כעת אנו נכנסים לאזור הטיפול בלייבלים:

• הפעולה הראשונה כאן היא IsLabelDeclaration, המקבלת מחרוזת ובודקת אם היא הכרזת לייבל או לא:

bool IsLabelDeclaration(char \*s)

{

//Checks whether a part of the text is a label declaration.

if (s[strlen(s) - 1] == ':')

{

//Removes the colon in order to save the label's name.

s[strlen(s) - 1] = '\0'; return true;

}

return false;

}

אם המחרוזת נגמרת בנקודותיים – היא הכרזה על לייבל, נמחק את הנקודותיים ונחזיר true. אם לא ,

.false נחזיר

• הפעולה הבאה ממירה את כתובת הלייבל לקוד בינארי )ההמרה היא לבינארי מפני שזו המרה יותר פשוטה, כשנקרא לפונקציה זו נמיר את הפלט מבינארי להקסדצימלי באמצעות הפעולה

:)BinToHex

void LabelAddressToBinary(char \*label, char \*\*\*labels, char \*converted)

{

//Gets the label's address by its name, and converts it to binary.

char decPlace[5]; int err;

if (label[strlen(label) - 1] == '\n')

//Cuts the \n in case the label name comes with it.

label[strlen(label) - 1] = '\0';

for (int i = 0; i < memorySize; i++) if (!strcmp(labels[i], label))

err = \_itoa\_s(i, decPlace, 5, 10);

DecimalToBinary(decPlace, converted, 16);

}

אם קוד ה-imm הגיע עם ירידת שורה בתו האחרון, מורידים אותו – ולאחר מכן מתבצע חיפוש אחר המחרוזת במערך הלייבלים. כשמגיעים לערך i הרצוי ממירים אותו למחרוזת דצימלית בעזרת הפעולה

\_itoa\_s, וממירים לבינארי באמצעות הפעולה שכתבנו קודם.

• הפעולה הבאה היא פעולה המוסיפה תווים למחרוזות שאורכן לא מספיק, ושמה AddToEight:

char \*AddToEight(char \*hex)

{

//Adds '0' bits to a hexadecimal code's beginning if it is not 8 characters long. char \*afterAddition[outputLineLength+1];

for (unsigned int i = 0; i < outputLineLength - strlen(hex); i++)

afterAddition[i] = '0';

afterAddition[outputLineLength - strlen(hex)] = '\0'; strcat\_s(afterAddition, outputLineLength + 1, hex);

return afterAddition;

}

קיימת לולאה המוסיפה אפסים למחרוזת afterAddition שהגדרנו, כמות האפסים נקבעת לפי אורך מחרוזת הקלט )האורך הדרוש פחות אורך הקלט.( לאחר מכן באמצעות פעולת strcat\_s נוסיף לאפסים את מחרוזת הקלט ונחזיר את afterAddition.

• הפעולה הבאה היא פעולה המטפלת בפקודת .word :

void HandleWord(char \*value, char \*place, char \*fullCode)

{

//Converts of the value string to hexadecimal. char \*fullLine = (char\*)malloc(sizeof(char) \* 33);

fullLine[0] = '\0'; if (strlen(value) > 2) if (value[0] == '0'&&value[1] == 'x') if (strlen(value) < 10)

{

fullLine = AddToEight(value + 2);

}

else

fullLine = value + 2;

if (fullLine[0] == '\0')

{

DecimalToBinary(value, fullLine, 32); BinToHex(fullLine, value);

fullLine = value;

}

//Adds the hexadecimal value to the overall output. if (strlen(fullCode) <= atoi(place) \* outputLineLength)

{

for (int i = strlen(fullCode); i < atoi(place) \* (outputLineLength+1); i++)

if (i % 9 != 8) fullCode[i] = '0';

else

fullCode[i] = '\n';

}

fullCode[(atoi(place)) \* (outputLineLength + 1) - 1] = '\n'; for (int i = 0; i < outputLineLength; i++)

fullCode[atoi(place) \* (outputLineLength+1) + i] = fullLine[i]; fullCode[(atoi(place) + 1) \* (outputLineLength+1)-1] = '\0';

}

באופן כללי הפעולה מקבלת את מיקום ההדפסה בזיכרון, המספר שנדרש להדפיס ואת המחרוזת המייצגת את הזיכרון, הגדרנו משתנה fullLine המייצג את השורה עליה נעבוד. מתבצעת בדיקה אם הערך value הוא הקסדצימלי – אם כן, נדאג שיהיה באורך שורה ונכניס לערך fullLine, אם לא, נמיר להקסדצימלי ונכניס לערך fullLine. אם אורך הזיכרון לא מגיע לשורה אליה מכניסים את הערך, נמלא אותו באפסים עד לשם ונוסיף בסוף את השורה הרצויה, אם לא, פשוט נחליף את המקומות בזיכרון בשורה שאליה מפנה הכתובת ונכניס לשם את המחרוזת fullLine.

**Main**

נסביר את התכנית הראשית בחלקים: 1(

int main(int argc, char \*argv[])

{

if (argc != 4)

exit(1);

if (strcmp(argv[1], "asm"))

exit(1);

FILE \*stream, \*output;

errno\_t err = fopen\_s(&stream, argv[2], "r");

if (stream == 0) {

fprintf\_s(stderr, "Error opening program.asm");

exit(1);

}

char line[maxLineLength+1]; char \*context = NULL; char \*\*labels = (char\*\*)malloc(sizeof(char) \* maxLabelLength \* memorySize); //Allocates memory to the labels array. for (int i = 0; i < memorySize; i++) labels[i] = (char\*)malloc(sizeof(char) \* maxLabelLength);

int i = 0;

בתחילת הפונקציה בדקנו אם הקלט תקין, אם כמות הארגומנטים לא נכונה או הארגומנט הראשון אינו ”asm“ נצא מהתוכנה. לאחר מכן נפתח את קובץ הקלט ונגדיר ערכים שנשתמש בהם בהמשך: line – מחרוזת השומרת את התוכן של שורה בקלט. context – מחרוזת שבה נשתמש בפעולת strtok לשמירת הריצות הקודמות של הפעולה. labels – מערך השומר את שמות הלייבלים במקומות אליהם הם מפנים, נאתחל אותו בלולאה. i – ערך מספרי השומר את מספר השורה, על מנת שנוכל לשמור לייבלים במקומות המתאימים במערך.

2( לאחר מכן נריץ לולאה הקוראת את הקובץ, ושומרת לייבלים במערך:

while (fgets(line, maxLineLength, stream) != NULL)

{

//Saves all labels inside the labels array, in the spot of the line they are in.

RemoveComment(line);

char \*possibleLabel = strtok\_s(line, " ,\n\t", &context); if (possibleLabel != NULL)

{

if (IsLabelDeclaration(possibleLabel))

{

strcpy\_s(labels[i], maxLabelLength, possibleLabel); if (strtok\_s(NULL, " ,\n\t", &context) != NULL)

i++;

}

else i++;

}

}

fclose(stream);

בתוך הלולאה נמחק את ההערה משורת הקלט, ונחלק אותה לחלקים על ידי strtok\_s, אם אחד החלקים עובר את בדיקת הפעולה IsLabelDeclaration, זה אומר שהוא לייבל, ונשמור אותו במערך. 3(

//Reopens the input file to read from the beginning. err = fopen\_s(&stream, argv[2], "r");

if (stream == 0) {

fprintf\_s(stderr, "Error opening program.asm");

exit(1);

}

char \*outputText = (char\*)malloc(sizeof(char)\*(memorySize \* (outputLineLength +

1)) + 1); outputText[0] = '\0';

//Defines the string to which the line of code is converted. char \*hexCode[outputLineLength+1];

while (fgets(line, maxLineLength, stream) != NULL)

{

hexCode[0] = '\0';

//Removes comments from the line.

RemoveComment(line);

char \*part = strtok\_s(line, " ,\n\t", &context);

לאחר מכן פתחנו את הקובץ לקריאה מחדש, לשם המרתו לקוד הקסדצימלי, בקטע הנ"ל מופיעה הגדרת מחרוזת הפלט outputText, הגדרנו בתוך הלולאה hexCode – מחרוזת הכוללת את הקוד המומר של השורה הספציפית s. נוריד את ההערה משורת האסמבלי ונגדיר part – מחרוזת הכוללת את המילה הראשונה הרלוונטית בשורה.

4( נסביר את שאר הלולאה:

if (part != NULL)

{

//If there is no label, the command is converted and added.

if (!IsLabelDeclaration(part))

{

strcpy\_s(hexCode,outputLineLength+1,TextToHex(part));

}

//If the command is ".word", we skip the line we will handle it in the next reading.

if (part[0] == '.')

continue;

}

while (part != NULL)

{

if (!IsLabelDeclaration(part))

{

part = strtok\_s(NULL, " ,\t\n", &context);

if (part != NULL)

{

//If the part is a number, it is converted to hexadecimal and added to the overall code.

if ((part[1] == 'x'&&part[0] == '0') || (part[0] >=

'0'&&part[0] <= '9') || part[0] == '-')

{

strcat\_s(hexCode, outputLineLength + 1,

ImmToHex(part));

part = NULL;

}

//If the part starts with a letter, and is not a known command, it is a label and is added as its address to the code.

else if (((part[0] >= 'a'&&part[0] <= 'z') || (part[0]

>= 'A'&&part[0] <= 'Z')) && !strcmp(part, TextToHex(part)))

{

char addition2[17];

addition2[0] = '\0';

LabelAddressToBinary(part, labels, addition2);

addition2[16] = '\0'; char addition3[5]; addition3[0] = '\0';

BinToHex(addition2, addition3);

strcat\_s(hexCode, outputLineLength + 1, addition3);

}

//If it is neither, it is added to the code.

else

{

char \*newHexCode[outputLineLength + 1]; strcpy\_s(newHexCode, outputLineLength + 1, hexCode);

strcat\_s(newHexCode, outputLineLength + 1,

TextToHex(part));

strcpy\_s(hexCode, outputLineLength + 1, newHexCode);

}

}

}

}

if (hexCode[0] != '\0'&&hexCode[0] != '\n')

{

strcat\_s(outputText, memorySize \* outputLineLength + 1, hexCode); strcat\_s(outputText, memorySize \* outputLineLength + 1, "\n");

} i++;

}

בהתחלה מתבצעת בדיקה על המילה הראשונה בשורה, אם זהו לא לייבל הוא מומר על ידי TextToHex ומתווסף ל-hexCode. אם המילה מתחילה בנקודה, זוהי פקודת word ובה נתעסק אחר כך. לאחר מכן נעבור על כל המילים שנשארו בשורה עם לולאה פנימית, שהתנאי שלה הוא שהפלט של strtok\_s לא יהיה NULL – כלומר, השורה עוד לא נגמרה. אם מילה היא מספר הקסדצימלי/דצימלי ,הוא מתווסף לקוד על ידי פקודת ImmToHex, אם המילה מתחילה באות, ולא נכללת במילים השמורות )כלומר הפלט של TextToHex שווה לקלט(, זה אומר שזו קריאה ללייבל, ושם נציב את כתובתו שנמצא תוך חיפוש במערך labels. אם החלק לא נכלל באף אחת מהקבוצות הללו, זו מילה שמורה ונוסיף את המרתה לקוד של השורה. לבסוף, אם hexCode לא ריק, ולא מכיל רק ירידת שורה תוכנו מתווסף ל-

.outputText

)5

fclose(stream);

err = fopen\_s(&stream, argv[2], "r");

if (stream == 0) {

fprintf\_s(stderr, "Error opening program.asm");

exit(1);

}

while (fgets(line, maxLineLength, stream) != NULL)

{

//Handles the .word command using the method and the text that comes after it in the line.

char \*part = strtok\_s(line, " ,\n\t", &context); if (part != NULL)

if (part[0] == '.')

HandleWord(strtok\_s(NULL, " ,\n\t", &context), strtok\_s(NULL,

" ,\n\t", &context), outputText);

}

//Prints the hexadecimal code to the output file.

err = fopen\_s(&output, "memin.txt", "w");

if (stream == 0) {

fprintf\_s(stderr, "Error opening mem.txt");

exit(1);

}

fprintf\_s(output, "%s", outputText);

fclose(stream); fclose(output);

for (int i = 0; i < memorySize; i++) if (labels[i] != NULL)

free(labels[i]);

free(labels);

free(outputText);

return 0;

}

לאחר הלולאה סוגרים את קובץ הקלט, ופותחים לקריאה מחדש על מנת לבצע את פעולות ה-word. בלולאה שוב עוברים על שורות הקובץ, אם הפקודה מתחילה בנקודה זו פקודת word, ואז מפנים לפעולה HandleWord. לבסוף פותחים את קובץ הפלט, מדפיסים אליו את outputText, סוגרים את הקבצים, משחררים את המחרוזות והתוכנה מסיימת את ריצתה.

# סימולטור

מטרת התוכנית: מימוש התוכנית המתוארת בשפת מכונה )הוראות מפורטות בקובץ מטלת הפרויקט(.

**תחילת הקוד** ייבוא ספריות והגדרת קבועים:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <math.h>

#include <stdbool.h>

#define numOfRegisters 16

#define memorySize 65536 #define wordSize 8

numOfRegisters מייצג את כמות הרגיסטרים, memorySize מייצג את כמות השורות בזיכרון, wordSize זהו אורך שורה בקבצי הקלט והפלט.

**פעולות** הפעולה הבאה היא פעולת המרה מקוד הקסדצימלי לקוד בינארי:

void HexToBin(char \*str, char \*binString)

{

//Converts a word from hexadecimal to binary. binString[0] = '\0';

for (unsigned int i = 0; i < strlen(str); i++)

{

if (str[i] == '0')

strcat\_s(binString, strlen(binString) + 5, "0000"); if (str[i] == '1')

strcat\_s(binString, strlen(binString) + 5, "0001"); if (str[i] == '2')

strcat\_s(binString, strlen(binString) + 5, "0010"); if (str[i] == '3')

strcat\_s(binString, strlen(binString) + 5, "0011"); if (str[i] == '4')

strcat\_s(binString, strlen(binString) + 5, "0100"); if (str[i] == '5')

strcat\_s(binString, strlen(binString) + 5, "0101"); if (str[i] == '6')

strcat\_s(binString, strlen(binString) + 5, "0110"); if (str[i] == '7')

strcat\_s(binString, strlen(binString) + 5, "0111"); if (str[i] == '8')

strcat\_s(binString, strlen(binString) + 5, "1000"); if (str[i] == '9')

strcat\_s(binString, strlen(binString) + 5, "1001");

if (str[i] == 'A')

strcat\_s(binString, strlen(binString) + 5, "1010"); if (str[i] == 'B')

strcat\_s(binString, strlen(binString) + 5, "1011"); if (str[i] == 'C')

strcat\_s(binString, strlen(binString) + 5, "1100"); if (str[i] == 'D')

strcat\_s(binString, strlen(binString) + 5, "1101"); if (str[i] == 'E')

strcat\_s(binString, strlen(binString) + 5, "1110"); if (str[i] == 'F')

strcat\_s(binString, strlen(binString) + 5, "1111");

}

{הפעולה עוברת על כל תו במחרוזת str ומוסיפה ל-binString ארבעה תוים לפי ערכו. הפעולה לא מחזירה פלט אלא עורכת מחרוזת קלט.

**Main**

כעת נעבור על הפעולה main: 1(

int main(int argc, char \*argv[])

{

errno\_t err; if (argc != 7)

exit(1);

if (strcmp(argv[1], "sim"))

exit(1);

//Creates an array of registers of size 16 int registers[numOfRegisters]; for (int i = 0; i < numOfRegisters; i++)

{

registers[i] = 0;

}

//Creates word of zero. char zeroWord[wordSize]; for (int i = 0; i < wordSize; i++)

{

zeroWord[i] = '0';

}

int PC = 0; int count = 0;

בתחילת הפעולה מתבצעת בדיקה על ארגומנטי הקלט, אם כמות הארגומנטים לא נכונה או הארגומנט הראשון לא שווה ל-”sim“, התוכנה נסגרת. הגדרנו מערך לרגיסטרים, מילה שמייצגת שורה ריקה

)מלאה באפסים( ואת המשתנים PC ו-count הדרושים לקובצי הפלט.

)2

//Reads the memory- memin.txt

FILE \*meminFilePoint = NULL; err = fopen\_s(&meminFilePoint, argv[2], "r");

if (meminFilePoint == 0) {

fprintf\_s(stderr, "Error opening memin.txt");

exit(1);

}

//Creates an array of the memory. char memory[memorySize][wordSize + 1];

for (int i = 0; i < memorySize; i++)

{

for (int j = 0; j < wordSize; j++)

{

memory[i][j] = '0';

}

}

char s[wordSize + 1];

for (int i = 0; i < memorySize && fgets(s, wordSize + 1, meminFilePoint) != NULL; i++)

{

if (strcmp(s, "\n") == 0)

{

i--;

}

else

{

for (int j = 0; j < wordSize; j++)

{

memory[i][j] = s[j];

}

}

}

fclose(meminFilePoint);

בחלק זה אנו פותחים את קובץ ה-memin, מגדירים מערך לזיכרון וממלאים אותו לפי קובץ הקלט.

3( לפני ריצת הלולאה, מגיע הקוד הבא:

FILE \*traceFilePoint;

err = fopen\_s(&traceFilePoint, argv[5], "w");

if (traceFilePoint == 0) {

fprintf\_s(stderr, "Error opening trace.txt");

exit(1);

}

char fullInstHexa[wordSize + 1]; char fullInstBinar[wordSize \* 4 + 1];

בו פתחנו את הקובץ trace לכתיבה, והגדרנו מחרוזות שיכללו את הקוד של שורה כלשהי בבינארי והקסדצימלי. 4(

כעת נסביר את הלולאה הרצה על הזיכרון:

//an infinite loop which stops at halt and executes one instruction in each loop.

while (PC < memorySize)

{

count++;

//reading the row number PC for (int i = 0; i < wordSize; i++)

{

fullInstHexa[i] = memory[PC][i];

}

fullInstHexa[wordSize] = '\0'; fullInstBinar[0] = '\0';

HexToBin(fullInstHexa, fullInstBinar);

//dividing it to parts int instOpcode = 0; int instRd = 0; int instRs = 0; int instRt = 0; int instImm = 0;

for (int i = 0; i < 4; i++)

{

instOpcode += (int)(fullInstBinar[i] - '0') \* (int)pow(2, 3 - i); instRd += (int)(fullInstBinar[i + 4] - '0') \* (int)pow(2, 3 - i); instRs += (int)(fullInstBinar[i + 8] - '0') \* (int)pow(2, 3 - i); instRt += (int)(fullInstBinar[i + 12] - '0') \* (int)pow(2, 3 - i);

}

instImm = -1 \* ((int)(fullInstBinar[16] - '0') \* (int)pow(2, 15));

for (int i = 0; i < 15; i++)

{

instImm += (int)(fullInstBinar[17 + i] - '0') \* (int)pow(2, 14 - i);

}

;'0\'fullInstBinar[32] = בתחילת הלולאה מעדכנים את ערך count, טוענים את מיקום הזיכרון הרלוונטי למחרוזת fullInstHex, ממירים לבינארי ולאחר מכן מפרידים לפקודה ,3 ערכים ו-imm.

5( בחלק הבא אנו מדפיסים את ה-trace לקובץ הרלוונטי:

//adding a line in trace.txt

char traceNextLineCoding[(numOfRegisters + 2)\*(wordSize + 1) + 2]; traceNextLineCoding[(numOfRegisters + 2)\*(wordSize + 1)] = '\0'; for (int i = 0; i < (numOfRegisters + 2); i++)

{

char temporary[wordSize + 1];

temporary[wordSize] = '\0';

if (i == 0)

{

snprintf(temporary, wordSize + 1, "%lX", PC);

} else if (i == 1)

{

for (int j = 0; j < wordSize; j++)

{

temporary[j] = fullInstHexa[j];

}

}

else

{

snprintf(temporary, wordSize + 1, "%lX", registers[i - 2]);

}

int len = wordSize - strlen(temporary); // get the missing part size memmove(temporary + len, temporary, strlen(temporary)); // move the string characters to the end

for (int i = 0; i < len; i++) // fill the beginning characters with

'0'

temporary[i] = '0';

temporary[wordSize] = '\0';

for (int j = 0; j < wordSize; j++)

{

traceNextLineCoding[i\*(wordSize + 1) + j] = temporary[j];

}

traceNextLineCoding[(i + 1)\*(wordSize + 1) - 1] = ' ';

}

traceNextLineCoding[(wordSize + 1)\*(numOfRegisters + 2)] = '\n'; traceNextLineCoding[(wordSize + 1)\*(numOfRegisters + 2) + 1] = '\0'; fprintf(traceFilePoint, traceNextLineCoding); שורת הפקודה ,PC (נגדיר מחרוזת שתכלול את הפלט, ונכניס אליה ערכים לפי הפורמט המתבקש ).והרגיסטרים

)6

//executing according to the opcode

bool exit = false;

switch (instOpcode)

{

case 0:

//add: R[rd]=R[rs]+R[rt]

registers[instRd] = registers[instRs] + registers[instRt]; break; case 1:

//sub: R[rd]=R[rs]-R[rt]

registers[instRd] = registers[instRs] - registers[instRt]; break; case 2:

//and: R[rd]=R[rs]&R[rt]

registers[instRd] = registers[instRs] & registers[instRt]; break; case 3:

//or: R[rd]=R[rs]|R[rt]

registers[instRd] = registers[instRs] | registers[instRt]; break; case 4:

//sll: R[rd]=R[rs]<<R[rt]

registers[instRd] = registers[instRs] \* (int)pow(2, registers[instRt]); break; case 5:

//sra: R[rd]=R[rs]>>R[rt]

registers[instRd] = registers[instRs] / (int)pow(2, registers[instRt]); break; case 6:

//limm: R[rd] = sign extended imm (bits 31:16=sign, bits 15:0=imm) registers[instRd] = instImm; break; case 7:

//beq: if (R[rs] == R[rt]) pc = imm-1

if (registers[instRs] == registers[instRt])

PC = instImm - 1;

break; case 8:

//bgt: if (R[rs] > R[rt]) pc = imm-1

if (registers[instRs] > registers[instRt])

PC = instImm - 1; break; case 9:

//ble: if (R[rs] <= R[rt]) pc = imm-1 if (registers[instRs] < registers[instRt])

PC = instImm - 1; break; case 10:

//bne: if (R[rs] != R[rt]) pc = imm-1 if (registers[instRs] != registers[instRt])

PC = instImm - 1; break; case 11:

//jal: R[15] = pc + 1 (next instruction address), pc = imm-1

registers[numOfRegisters - 1] = PC + 1;

PC = instImm - 1;

break; case 12:

{

//lw: R[rd] = MEM[R[rs]+imm] char temporary[wordSize + 1]; temporary[wordSize] = '\0';

for (int i = 0; i < wordSize; i++)

{

temporary[i] = memory[registers[instRs] + instImm][i];

}

registers[instRd] = (int)strtoul(temporary, NULL, 16);

break;

} case 13:

{

//sw: MEM[R[rs]+imm] = R[rd] char temporary[wordSize] = { '\0' };

snprintf(temporary, wordSize + 1, "%lX", registers[instRd]);

for (int i = 0; i < wordSize; i++)

{

memory[registers[instRs] + instImm][i] = temporary[i];

}

break;

}

case 14:

//jr: pc = R[rd]-1

PC = registers[instRd] - 1;

break; case 15:

//halt: Halt execution, exit simulator

exit = true;

break;

} if (exit == true)

break;

PC++;

registers[0] = 0;

} fclose(traceFilePoint);

בסוף הלולאה מתבצעת הפעולה המוגדרת בשורת הזיכרון על ידי שימוש ב-switch. פקודת halt גוררת יציאה מהלולאה על ידי המשתנה PC ,exit גדל באחד והרגיסטר האפס מוגדר לאפס )למקרה שהמשתמש שינה אותו באופן לא חוקי.(

7( בסוף הקוד נפתחים קבצי הפלט )מלבד trace שנפתח כבר( לכתיבה:

FILE \*memoutFilePoint;

err = fopen\_s(&memoutFilePoint, argv[3], "w");

if (memoutFilePoint == 0) {

fprintf\_s(stderr, "Error opening memout.txt");

exit(1);

}

FILE \*regoutFilePoint;

err = fopen\_s(&regoutFilePoint, argv[4], "w");

if (regoutFilePoint == 0) {

fprintf\_s(stderr, "Error opening regout.txt");

exit(1);

}

FILE \*countFilePoint;

err = fopen\_s(&countFilePoint, argv[6], "w");

if (countFilePoint == 0) {

fprintf\_s(stderr, "Error opening count.txt");

exit(1);

}

//creating register array file - regout.txt

for (int i = 0; i < numOfRegisters; i++)

{

char regoutNextLineCoding[wordSize + 2]; snprintf(regoutNextLineCoding, wordSize + 1, "%lX", registers[i]); int len = wordSize - strlen(regoutNextLineCoding); // get the missing part size

memmove(regoutNextLineCoding + len, regoutNextLineCoding, strlen(regoutNextLineCoding)); // move the string characters to the end for (int i = 0; i < len; i++) // fill the beginning characters with '0' regoutNextLineCoding[i] = '0'; regoutNextLineCoding[wordSize] = '\n'; regoutNextLineCoding[wordSize + 1] = '\0';

fprintf(regoutFilePoint, regoutNextLineCoding);

}

//creating file memout.txt - the memory after the simulator

for (int i = 0; i < memorySize; i++)

{

char temporary[wordSize + 2]; memcpy(temporary, &memory[i], wordSize); temporary[wordSize] = '\0';

int len = wordSize - strlen(temporary); // get the missing part size memmove(temporary + len, temporary, strlen(temporary)); // move the string characters to the end

for (int j = 0; j < len; j++) // fill the beginning characters with '0'

temporary[j] = '0';

temporary[wordSize] = '\n';

temporary[wordSize + 1] = '\0';

fprintf(memoutFilePoint, temporary);

}

for (int i = 0; i < memorySize; i++)

memory[i][wordSize] = '\0';

//creating count.txt - how much instructions were executed

fprintf(countFilePoint, "%d\n", count);

//closing files fclose(memoutFilePoint); fclose(regoutFilePoint);

fclose(countFilePoint);

return 0;

{ קודם כל פותחים את כל הקבצים לכתיבה, בלולאה הראשונה ממירים את תכני הרגיסטרים למחרוזות הקסדצימליות ומדפיסים לקובץ regout. בלולאה השניה מדפיסים את תוכן הזיכרון לקובץ memout, ולבסוף, מדפיסים את הערך count לקובץ count. סוגרים את כל הקבצים והתוכנה מסיימת את ריצתה.

# לוח הכפל

מטרת הקוד : כתיבת לוח הכפל:

mem[1024+(j-1)\*10+i-1] , 1 ≤ 𝑖, 𝑗 ≤ 10

קוד התכנית מתחיל בהגדרת מספר ערכי התחלה- ערכי הstack וערכי static אחרים: Main:

limm $sp, $zero, $zero, 2048 #initialization of the stack address limm $s0, $zero, $zero, 1024 #saving in address limm $s1, $zero, $zero, 9 #size of the mul table-1 limm $s2, $zero, $zero, 1 #constant 1

כלומר: sp - הstack מאותחל לכתובת ראשונית של 2048

S0 - מאותחל לכתובת ההתחלה אליה יש לכתוב את לוח הכפל

S1 - מייצג את גודל לוח הכפל פחות 1 S2 - קבוע 1

לאחר מכן, מתבצעות שני לולאות- לולאה בתוך לולאה, אשר רצות על ערכי 1-i מ0 עד 9 )לולאה חיצונית(, ועל

ערכי 1-j מ0 עד 9 )לולאה פנימית.(

* הגדלות 1-i-1, j רשומות באדום
* הלולאות, ותנאי היציאה מהן מסומנים בירוק • הקטע המודגש בכחול (...) מתבצע עבור ערכי 1-i-1, j השונים, ומוסבר בהמשך

limm $a0, $zero, $zero, 0 #initialization of i-1 OutLoop:

bgt $zero, $a0, $s1, EndOutLoop #if i>10 out limm $a1, $zero, $zero, 0 #initialization of j-1 InLoop: bgt $zero, $a1, $s1, EndInLoop #if j>10 out

...

add $a1, $a1, $s2, 0 #j++ beq $zero, $zero, $zero, InLoop EndInLoop:

add $a0, $a0, $s2, 0 #i++ beq $zero, $zero, $zero, OutLoop EndOutLoop: :לבסוף, התכנית מסתיימת

halt $zero, $zero, $zero, 0

הקטע המודגש בכחול(...)

1. jal $zero, $zero, $zero, Mul add $t0, $zero, $v0, 0 #$t0=i\*j

1. jal $zero, $zero, $zero, PrintAddress
2. sw $t0, $v0, $zero, 0 #mem[1024+(j-1)\*10+i-1]=i\*j

קטע זה מתבצע עבור כל 1-i-1, j ומבצע את הפעולה :

1[-mem[1024+(j-1)\*10+iהקטע מחולק לשלושה חלקים:

1(ביצוע הפעולה :

$t0=i\*j

2(ביצוע הפעולה :

$v0=1024+(j-1)\*10+i-1

3( השמירה לזכרון:

mem[1024+(j-1)\*10+i-1]= i\*j :1,2 נפרט על חלקים

1(חישוב i\*j מתבצע בפעולה הבאה:

תחילה מתבצע שמירה לזכרון של הערכים אותם נרצה לאחר החזרה מהפונקציה:

Mul: #$a0=i-1, $a1=j-1

sub $sp, $sp, $s2, 0 sw $a0, $sp, $zero, 0

sub $sp, $sp, $s2, 0

sw $a1, $sp, $zero, 0 לאחר מכן מתבצע לולאה שתתרחש 1-j פעמים, ובה נוסיף לv0 את הערך i בכל פעם.

add $a0, $a0, $s2, 0 #$a0=i add $v0, $a0, $zero, 0 #$v0=i limm $t0, $zero, $zero, 1 #$t0=1 Loop1: bgt $zero, $t0, $a1, EndLoop1 add $v0, $v0, $a0, 0 add $t0, $t0, $s2, 0 #$t0++ beq $zero, $zero, $zero, Loop1 EndLoop1: כעת מתבצע שליפה בחזרה מהזכרון

lw $a0, $sp, $zero, 0 add $sp, $sp, $s2, 0 lw $a1, $sp, $zero, 0 add $sp, $sp, $s2, 0 )callerלבסוף חזרה לתכנית )ל

jr $ra, $zero, $zero, 0

2( ביצוע חלק זה מתבצע בעזרת הפעולה הבאה.

תחילה מתבצע שמירה לזכרון של הערכים אותם נרצה לאחר החזרה מהפונקציה:

PrintAddress: #$a0=i-1, $a1=j-1 sub $sp, $sp, $s2, 0

sw $a0, $sp, $zero, 0

sub $sp, $sp, $s2, 0

sw $a1, $sp, $zero, 0 לאחר מכן מתבצעת לולאה שתתרחש 1-j פעמים, שבה ייתווסף לv0, הערך 10 בכל פעם. סה"כ: 10\*)1-j(

limm $t1, $zero, $zero, 10 #constant 10 add $v0, $s0, $zero, 0 #$v0=1024 limm $t2, $zero, $zero, 1 #$t2=1 Loop2:

bgt $zero, $t2, $a1, EndLoop2 add $v0, $v0, $t1, 0 add $t2, $t2, $s2, 0 #$t2++ beq $zero, $zero, $zero, Loop2 EndLoop2: )i-1(\*1 :הערך 1 בכל פעם. סה"כ ,v0פעמים, שבה יתווסף ל i-1 לאחר מכן מתבצעת לולאה שתתרחש

limm $t2, $zero, $zero, 1 #$t2=1 Loop3:

bgt $zero, $t2, $a0, EndLoop3 add $v0, $v0, $s2, 0 add $t2, $t2, $s2, 0 #$t2++ beq $zero, $zero, $zero, Loop3 EndLoop3:

כעת מתבצע שליפה בחזרה מהזכרון

lw $a0, $sp, $zero, 0 add $sp, $sp, $s2, 0 lw $a1, $sp, $zero, 0 add $sp, $sp, $s2, 0 )callerלבסוף חזרה לתכנית )ל

jr $ra, $zero, $zero, 0

## Bubble sort

מטרת הקוד: סידור מערך בכתובות זכרון 1039-1024 בסדר יורד לפי אלגוריתם bubblesort.

הקוד מתחיל במספר הגדרות ואתחולים: Main:

limm $sp, $zero, $zero, 2048 limm $s0, $zero, $zero, 1 #constant 1 limm $s1, $zero, $zero, 1038 #constant (end address-1)

למעשה:

Sp- מוגדר כ2048. זוהי כתובת תחילת הstack

S0- קבוע של 1 s1 - כתובת סוף המערך פחות אחד

לאחר מכן מתבצעת לולאה בתוך לולאה. בכל לולאה חיצונית האיבר הקטן ביותר עובר לסוף המערך )של אותה לולאה(

)כבועה.(

הדבר מתבצע על ידי השוואה של כל איבר לאיבר הבא אחריו, ובמקרה הצורך מחליף ביניהם. דבר זה מתבצע בפעולה

הנקראת בכחול .הלולאות ותנאי הריצה עליהן מסומנים בירוק, ובאדום מתבצע עדכון איברי ריצת הלולאה

limm $t0, $zero, $zero, 1038 #a parameter holding the end of EachInLoop Run StartOutLoop:

limm $a0, $zero, $zero, 1024 #start address bgt $zero, $a0, $t0, EndOutLoop

StartInLoop:

bgt $zero, $a0, $t0, EndInLoop jal $zero, $zero, $zero, Bubble

add $a0, $a0, $s0, 0 #$a0++ beq $zero, $zero, $zero, StartInLoop EndInLoop: sub $t0, $t0, $s0 ,0 #$t0-- beq $zero, $zero, $zero, StartOutLoop EndOutLoop:

לבסוף התכנית מסתיימת

halt $zero, $zero, $zero, 0

הפעולה Bubble פעולה זו משווה בין האיבר בכתובת a0 לאיבר בכתובת לאחריו, ומחליפה ביניהם במקרה הצורך.

Bubble: #$a0= address which should be compared to the one next to it.

בתחילה, מתבצע שמירה לstack של האיברים אותם נצטרך בחזרה מהפעולה:

sub $sp, $sp, $s0, 0 #$sp-- sw $a0, $sp, $zero, 0 #keep $a0 for later

לאחר מכן אנו שולפים את האיברים בכתובת a0 ודבכתובת לאחריו )מסומן בסגול.(

add $a1, $a0, $s0, 0 #$a1=$a0+1 lw $t1, $a0, $zero, 0 #$t1=mem[$a0] lw $t2, $a1, $zero, 0 #$t2=mem[$a1]

נשווה בין הביטויים )בצהוב(, ובמקרה הצורך, נחליף ביניהם בשמירה בחזרה לזכרון )בכתום(.

bgt $zero, $t1, $t2, GoOn sw $t2, $a0, $zero, 0 #mem[$a0]=$t2 sw $t1, $a1, $zero, 0 #mem[$a1]=$t1 GoOn:

אז, מתבצע אחזור הרגיסטרים השונים מהstack:

lw $a0, $sp, $zero, 0 #keep $a0 for later add $sp, $sp, $s0, 0 #$sp++

ולבסוף, חוזרים לcaller:

jr $ra, $zero, $zero, 0

# Mergesort

מטרת הקוד: מיון מערך בזכרון בכתובות 1039-1024 בסדר יורד לפי אלגוריתם mergesort.

הקוד מתחיל בהגדרת מספר ערכים מאותחלים:

Main:

limm $sp, $zero, $zero, 2048 limm $s0, $zero, $zero, 1 #constant 1

limm $s1, $zero, $zero, 4096 #an address of a temporary help array limm $a0, $zero, $zero, 1024 #start address limm $a1, $zero, $zero, 1039 #end address

Sp- כתובת תחילת הstack

S0- קבוע 1

S1- כתובת בה יירשם מערך זמני לשם מימוש הפעולה

A0- כתובת תחילת המערך A1- כתובת סוף המערך

לאחר מכן הפעולה קוראת לפעולת mergesort המקבלים כתובת תחילה וסיום a0, a1 של מערך וממיינת אותו.

jal $zero, $zero, $zero, MergeSort :לאחר מכן הפעולה מסתיימת

halt $zero, $zero, $zero, 0

MergeSort MergeSort: #$a0 is the start address, $a1 is the end address

הפעולה היא פעולה רקורסיבית ובנויה מ5 חלקים: 1. במידה וכתובת התחילה והסיום הם אותה הכתובת המערך למעשה כבר ממוין:

bne $zero, $a0, $a1, GoOn #if ($a0=$a1)

jr $ra, $zero, $zero, 0 #end MergeSort

|  |  |
| --- | --- |
| GoOn: |  |
| שמירה בזכרון של ערכים שונים אותם נצטרך בחזרה מן הפעולה  sub $sp, $sp, $s0, 0 #$sp-- sw $ra, $sp, $zero, 0 #save $ra for later sub $sp, $sp, $s0, 0 #$sp-- sw $a0, $sp, $zero, 0 #save $a0 for later sub $sp, $sp, $s0, 0 #$sp-- sw $a1, $sp, $zero, 0 #save $a1 for later sub $sp, $sp, $s0, 0 #$sp--  sw $t0, $sp, $zero, 0 #save $t0 for later sub $sp, $sp, $s0, 0 #$sp-- sw $t1, $sp, $zero, 0 #save $t1 for later sub $sp, $sp, $s0, 0 #$sp-- sw $t2, $sp, $zero, 0 #save $t2 for later | .2 |
| .מציאת הכתובת אמצע במערך add $t0, $a0, $a1, 0 #$t0=$a0+$a1  sra $t0, $t0, $s0, 0 #$t0=($a0+$a1)/2 add $t1, $t0, $s0, 0 #$t1=($a0+$a1)/2 + 1 | .3 |
| מיון החלק הראשון של המערך )קריאה רקורסיבית עם כתובת סיום שהיא כתובת האמצע( | .4 |
| add $t2, $a1, $zero, 0 #$t2 is temporary for help  add $a1, $t0, $zero, 0 jal $zero, $zero, $zero, MergeSort |  |
| מיון החלק השני של המערך )קריאה רקורסיבית עם כתובת תחילית שהיא כתובת האמצע(  add $a1, $t2, $zero, 0 add $t2, $a0, $zero, 0 add $a0, $t1, $zero, 0 jal $zero, $zero, $zero, MergeSort  add $a0, $t2, $zero, 0 | .5 |
| כעת כאשר כל חצי של המערך ממוין, מתבצעים לולאות, להעברת הערכים מכל חצי מערך למערך זמני | .6 |

במיקום אחר בזכרון. העברת הערכים מתבצעת תוך השוואת ערכים מכל חלק של המערך. בסיום העברת הערכים מאחד החצאים, יועברו שאר הערכים מהחצי השני.

מסומן בצבעים:

אדום- הגדלת ערכים כחלק מהלולאות ירוק- הלולאות ותנאי היציאה מהן כחול- גישה לזכרון צהוב- שמירה לזכרון add $t2, $s1, $zero, 0 # an address of a temporary array StartLoop1:

bgt $zero, $a0, $t0, EndLoop1 bgt $zero, $t1, $a1, EndLoop1 lw $t3, $a0, $zero, 0 #$t3=mem[$a0] lw $v0, $t1, $zero, 0 #$v0=mem[$t1]

bgt $zero, $t3, $v0, caseB #if(mem[$a0]<=mem[$t1]) sw $v0, $t2, $zero, 0 #copy to temporary add $t1, $t1, $s0, 0 #$t1++ beq $zero, $zero, $zero, Continue

caseB: #else

sw $t3, $t2, $zero, 0 #copy to temporary add $a0, $a0, $s0, 0 #$t1++ Continue:

add $t2, $t2, $s0, 0 #$t2++ beq $zero, $zero, $zero, StartLoop1

EndLoop1:

bgt $zero, $a0, $t0, EndLoop2 lw $t3, $a0, $zero, 0 #copying to temporary sw $t3, $t2, $zero, 0 add $a0, $a0, $s0, 0 #$a0++ add $t2, $t2, $s0, 0 #$t2++

beq $zero, $zero, $zero, EndLoop1

EndLoop2:

bgt $zero, $t1, $a1, EndLoop3 lw $t3, $t1, $zero, 0 #copying to temporary sw $t3, $t2, $zero, 0

add $t1, $t1, $s0, 0 #$t1++ add $t2, $t2, $s0, 0 #$t2++ beq $zero, $zero, $zero, EndLoop2 :לבסוף מעתיק בחזרה את מערך העזר לכתובת הנכונה בזכרון .7

EndLoop3: #copy the temporary array

lw $a0, $sp, $zero, 4 #$a0 is the start address again add $t2, $s1, $zero, 0 # $t2 is again an address of a temporary array StartLoop4:

bgt $zero, $a0, $a1, EndLoop4 lw $t3, $t2, $zero, 0 #copying from temporary

sw $t3, $a0, $zero, 0 add $t2, $t2, $s0, 0 #$t2++ add $a0, $a0, $s0, 0 #$a0++ beq $zero, $zero, $zero, StartLoop4

EndLoop4:8. נשחזר לתוך הרגיסטרים את ערכיהם המקוריים מתוך הstack

lw $t2, $sp, $zero, 0 #upload $t2 for later

add $sp, $sp, $s0, 0 #$sp++

lw $t1, $sp, $zero, 0 #upload $t1 for later

add $sp, $sp, $s0, 0 #$sp++ lw $t0, $sp, $zero, 0 #upload $t0 for later

add $sp, $sp, $s0, 0 #$sp++

lw $a1, $sp, $zero, 0 #upload $a1 for later

add $sp, $sp, $s0, 0 #$sp++ lw $a0, $sp, $zero, 0 #upload $a0 for later add $sp, $sp, $s0, 0 #$sp++ lw $ra, $sp, $zero, 0 #upload $ra for later add $sp, $sp, $s0, 0 #$sp++ callerנצא מהפעולה, בחזרה ל .9

jr $ra, $zero, $zero, 0 #return