דוקומנטציה חיצונית לפרויקט במבנה המחשב

הסימולטור:

מבני הנתונים:

1. רשימה מקושרת לשמירת שורות trace בשם trace_line_list.
2. רשימה מקושרת לשמירת שורות hwtrace בשם hw_trace_line_list.
3. רשימה מקושרת לשמירת שורות לקובץ ה-Led בשם leds_trace_list.
4. רשימה מקושרת לשמירת שורות לקובץ dis7seg בשם dis7seg_trace_list.
5. רשימה מקושרת לשמירת אינטרפטי irq2 בשם irq2_list.
6. מטריצה 256X256 לשמירת נתוני המוניטור בשם monitor.
7. מערך של instruction (struct המכיל שדות של חלקי הפקודה) באורך 4096 לשמירת כל ה-instructions שמופיעים בקובץ imemin.txt, בשם imemin_instructions_array.
8. מערך באורך 4096 לשמירת הזכרון בשם dmem_array.
9. מטריצה 128X128 לשמירת נתוני הדיסק בשם disk_matrix.
10. מערך פוינטרים לרגיסטרים בשם reg_pointer_array.
11. מערך פוינטרים לרגיסטרי החומרה בשם IO_reg_pointer_array.

משתנים חשובים:

1. Cycle_counter שעוקב אחר הסייקלים.
2. משתנה לכל רגיסטר ולכל רגיסטר-חומרה.
3. PC.
4. מעקב אחר disk_timeout.
5. מעקב אחר סיגנל נוכחי curr_sig.
6. משתנה עבור ה-node הנוכחי בכל רשימה מקושרת.

פונקציות:

1. **פונקציות לפרסור קבצי קלט**
   a. Init_instructions_array
      i. קוראת מקובץ ה-imemin.txt (או שם אחר) ומאתחלת את המערך imemin_instructions_array.
   b. Init_dmem_array
      i. קוראת מקובץ ה-dmem.txt ומאתחלת את מערך dmem_array.
   c. Load_diskin
      i. קוראת מקובץ ה-diskin.txt, ומאתחלת את מטריצת disk_matrix.
   d. Load_irq2in
      i. קוראת מקובץ ה-irq2in.txt, ומאתחלת את הרשימה irq2_list.
      ii. פונקציית עזר ליצירת תא יחיד ברשימה והוספתו- add_irq2_node. מקבלת כארגומנט את מספר הסייקל.
   e. Load_files
      i. קוראת לכל יתר הפונקציות בסקשיין. אם הייתה שגיאה, יוצאת מהריצה.
2. **פונקציות ליצירת קבצי פלט**

a. Open_w_then_a, open_in_mode
   i. זוג פונקציות שמטרתן קלות בפתיחת קובץ חדש, באופן כזה שירסט את תוכנו אם היה קיים- ויאפשר להוסיף לו תוכן.
b. Convert_instruction_to_bits
   i. מקבלת struct של instruction, מחזירה את רצף הביטים המייצג אותו.
c. Find_limit_index_in_dmem, find_limit_index_in_diskout, find_limit_index_in_monitor
   i. פונקציות שמטרתן מציאת האינדקס שהחל ממנו הזכרון, הדיסק והמוניטור הם אפסים בלבד (כדי לא לכלול אותם בקובץ הפלט).
d. Create_dmemout_txt
   i. יצירת קובץ ה-dmemout.txt מתוך dmem_array.
e. Create_regout_txt
   i. יצירת קובץ ה-regout.txt מתוך ערכי הרגיסטרים בסיום.
f. Create_trace_txt
   i. יצירת קובץ ה-trace.txt מתוך הרשימה trace_line_list.
g. Create_hwregtrace_txt
   i. יצירת קובץ ה-hwregtrace.txt מתוך הרשימה hw_trace_line_list.
h. Create_cycles_txt
   i. יצירת קובץ ה-cycles.txt מתוך ערך המשתנה cycles_counter.
i. Create_leds_txt
   i. יצירת קובץ ה-leds.txt מתוך הרשימה leds_trace_list.
j. Create_display7seg_txt
   i. יצירת קובץ ה-display7seg.txt מתוך הרשימה display7seg_list.
k. Create_diskout_txt
   i. יצירת קובץ ה-diskout.txt מתוך המטריצה disk_matrix.
l. Create_monitor_txt
   i. יצירת קובץ ה-monitor.txt מתוך המטריצה monitor.
   ii. יצירת קובץ ה-monitor.yuv מתוך המטריצה monitor.
m. Create_out_files
   i. קוראת לכל פונקציות יצירת הקבצים.

3. פונקציות לניהול ומעקב הממשק עם הIO

a. Handle_cmds
   i. מבצעת בדיקה האם פקודת ה-out כתבה פקודה לדיסק או למוניטור, ומדמה את הביצוע של הפקודה בהתאם.
b. Io_commands_trace
   i. מבצעת מעקב אחר פעולות ה-IO באמצעות הרשימות המקושרות הרלוונטיות.
c. Add_hw_trace_node
   i. יוצרת ומוסיפה תא חדש לרשימה המקושרת hw_trace_line_list.
d. Add_leds_trace_node
   i. יוצרת ומוסיפה תא חדש לרשימה המקושרת leds_trace_list.

e. Add_dis7seg_trace_node
   i. יוצרת ומוסיפה תא חדש לרשימה המקושרת display7seg_list.

4. פונקציות המממשות instructions

a. Do_instruction_command(relevant registers) הוא המבנה הכללי, כאשר instruction זה פלייס הולדר לכל אחת מהפקודות המופיעות בטבלה. מקבלות כארגומנטים את מספרי הרגיסטרים הרלוונטיים לביצוען.
b. Commit_the_instruction(instruction)
   i. מקבלת struct של instruction, וקוראת לפונקציה המתאימה לביצועה. מחזירה 1 אם הצליחה, 1- אם קרתה שגיאה, ו-0 אם הפקודה היא Halt.

5. **פונקציות עזר ל-main**

a. Add_trace_node
   i. יצירה והוספת תא לרשימה המקושרת trace_node_list.
b. Copy_regs_array
   i. מחזירה מערך של סנפשוט הרגיסטרים באותה נקודת זמן.
c. Set_up_files
   i. מפרסרת ושומרת את כל ה-Pathים של הקבצים משורת הריצה.
d. Handle_ints
   i. מתמודדת עם אינטרפטים, מעדכנת את רגיסטרי ה-IO בהתאם.
e. Exec_instruction
   i. מריצה את commit_the_instruction. מסיימת את הריצה אם זו החזירה שגיאה. מעדכנת את cycle_counter, pc. מחזירה האם הפקודה היא halt.
f. Update_immediates
   i. מעדכנת את הרגיסטרים של Imm1, imm2 לערכים שניתנו ב-instruction.

6. Main

a. מטפלת בארגומנטי ההרצה
b. טוענת את קבצי הקלט
c. מבצעת לולאה של סייקל עד קבלת Halt \ שגיאה.
d. יוצרת את קבצי הפלט

פלואו של סייקל בודד:

1. מעקב והתמודדות עם אינטרפטים.
2. עדכון הרגיסטרים immediates.
3. הוספת תא עבור trace.
4. ביצוע הפקודה שניתנה (ועדכון כל יתר המשתנים ומבני הנתונים הנגזרים מכך).

<u>האסמבלר:</u>

1. **מבני נתונים ומשתנים חשובים:**

a. מערך בשם instruction_names ובו שמות הפעולות המפורטות במטלה, מסודרות באופן שכל אינדקס במערך מייצג את מספר הopcode של הפעולה.

b. מערך בשם register_names ובו שמות הרגיסטרים, מסודרים באופן שכל אינדקס במערך מייצג את מספר הרגיסטר המתאים.
c. משתנה בשם label_head המצביע לראש רשימה מקושרת של משתנים מסוג struct label_node המייצגים label בקוד האסמבלי. בכל צומת ברשימה זו ישמר שם ה-label והכתובת המתאימה לו בקוד האסמבלי.
d. מערך שלמים בגודל 4096 בשם data_memory_arr המייצג את הערכים שנשמרים בזיכרון הdata באמצעות פקודות word. . בהמשך, המידע הזה יועבר לקובץ הdmemin.

2. **פונקציות:**

a. find_first_word_start(char line[]) – הפונקציה מקבלת מצביע למחרוזת ומחזיקה את האינדקס בו מתחילה המילה הראשונהת כלומר את התו הראשון שאינו whitespace.
b. add_label(char name[], int address) – הפונקציה יוצרת אובייקט מסוג struct label_node ושומרת בו את הערכים המתקבלים כקלט. לאחר מכן, האובייקט מוכנס בתחילת הרשימה המקושרת עליה מפורט לעיל.
c. get_numerical_value(char str[]) – הפונקציה מקבלת מצביע למחרוזת המייצגת ערך שיוכנס לזיכרון. הפונקציה בודקת האם הקלט בפורמט דצימלי או הקסה-דצימלי וממירה את הערך למספר מסוג uint64_t בהתאם.
d. set_data_mem_arr(char address[], char data[]) – הפונקציה מקבלת שתי מחרוזות המייצגות כתובת וערך לשמירה בdata memory. הפונקציה מחשבת ערך מספרי עבור שתי המחרוזות (ע"י שימוש בפונקציה הקודמת) ומשימה את הערך בכתובת המתאימה במערך.
e. set_data_mem_file(FILE* data_mem) – הפונקציה מקבלת קובץ המייצג data memory, וכותבת בו את הערכים השמורים במערך המייצג זיכרון זה. כל תא במערך יועתק לשורה בקובץ בפורמט הקסה-דצימלי.
f. first_pass(FILE* input_file, FILE* data_mem) – הפונקציה המרכזית עבור המעבר הראשון על קובץ האסמבלי, משתמשת בכל הפונקציות שתוארו עד כה. הפונקציה מחזירה מונה המייצג את כתובת השורה הנקראת. הפונקציה עובדרת בלולה שורה-שורה על קובץ האסמבלי ומתייחסת לכל שורה לפי המקרים הבאים:
   - אם השורה מכילה רק רווחים או מתחילה בסימן # (שורת הערה), נתעלם ממנה.
   - אם השורה מציינת שם label (מילה מסתיימת בסימן ':'), נוסיף את הlabel לרשימה המקושרת, כאשר כתובתו תהיה ערכו הנוכחי של המונה.
   - אם השורה מתחילה במילה השמורה "word.", נכניס את המידע הרלוונטי לכתובת הרלוונטית בזכרון, בעזרת הפונקציות שתוארו לעיל.
   - אחרת, השורה היא שורת פקודה רגילה. לא נבצע דבר מלבד להגדיל את ערכו של המונה באחד. זהו המקרה היחיד בו נגדיל את המונה מכיוון ששאר המקרים אינם מייצגים שורה אמיתית שתופיע לבסוף בinstruction memory ולכן לא יתפסו שורה בקובץ הפלט הרלוונטי.

g. get_opcode(char str[]), get_reg(char str[]) – הפונקציות מקבלות מחרוזת המייצגת שם פקודה/רגיסטר ומחזירות בהתאמה את מספר הפקודה/רגיסטר המתאים. הן עושות זאת ע"י מעבר על המערך המתאים והחזרת האינדקס שבו המידע זהה לקלט.
h. get_next_word(char line[], int i, char word[]) – פונקציית עזר שמשמשת לקריאת מילה אחת מתוך המחרוזת line החלק מהתו ה-i לתוך המחרוזת word. לפני תחילת הקריאה נדלק על רווחים ופסיקים. נפסיק את הקריאה ברגע שנגיע לרווחת פסיקת או '#'. נחזיר את האינדקס שבו הפסקנו לקרוא במחרוזת הקלט.
i. get_label_address(char name[]) – הפונקציה תחזיר את הכתובת המתאימה לשם הlabel שהיא מקבלת. היא עושה זאת ע"י מעבר רשימת הlabels המקושרת. מתוך ההנחה שקובץ הקלט תקין, ההחזרה תבוצע רק כאשר יימצא הlabel המתאים.
j. get_immediate(char str[]) – הפונקציה מקבלת מחרוזרת המייצגת ערך immediate, ומחזירה את ערכו המספרי. הקלט יכול להיות שם label, או מספר (דצימלי או הקסה-דצימלי). הפונקציה מפרידה בין מקרים אלו ע"י בדיקת התו הראשון ומשתמשת בפונקציית עזר מתאימה (מבין אלה שתוארו בסעיפים c,i) להחזרת הקלט.
k. second_pass(FILE* input_file, FILE* inst_mem) – הפונקציה המרכזית עבור המעבר השני על קובץ האסמבלי. במעבר זה, נתעלם משורות שמייצגות הערות, הגדרות label, והכנסות לdata memory שכן טיפלנו במקרים אלה במעבר הראשון. נטפל רק בשורות המייצגות פקודות אסמבלי. נאתחל משתנה המייצג פקודה כערך מספרי, ונבצע: נמיר את המילה הראשונה (שם הפקודה) לopcode המתאים ונכתוב אותו ב6 הביטים העליונים. לאחר מכן את ארבעת המילים הבאות נמיר למספר הרגיסטר המתאים ונכתוב אותם בהתאם בארבעת הביטים הבאים הבאים בכל פעם. לבסוף נמיר את שתי המילים הבאות לimmediate המתאימים ונכתוב אותם בהתאם ב12 הביטים האחרונים בכל פעם. בסיום קריאת כל שורה כזו, נכתוב את הערך המספרי שבנינו כשורה בקובץ הInstruction memory בפורמט של 8 ספרות הקסה-דצימליות כנדרש. את החישובים וההמרות שתוארו נמצע באמצעות פונקציות העזר שתוארו לעיל.

3. **פלואו כללי:**

ראשית, נבדוק שסופקו מספר משתנים כמצופה ונפתח את הקבצים שהתקבלו כקלט (קובץ האסמבלי יפתח לקריאה, שני קבצי הפלט יפתחו לקריאה). נבדוק שהקבצים נפתחו באופן תקין ונמשיך למעבר הראשון על קובץ האסמבלי. לאחר המעבר הראשון, המערך שמייצג data memory מעודכן לכן נעביר את המידע ששמרנו בו לקובץ הפלט המייצג data memory כבר בשלב זה. נבצע rewind לקובץ האסמבלי ונעבור עליו בפעם השנייה בעזרת קריאה לפונקציה המתאימה. לאחר המעבר השנית גם קובץ הפלט המייצג instruction memory כבר מעודכן, לכן לאחר שלב זה נסגור את שלושת הקבצים ונסיים את ריצת התוכנית.

<u>הטסטים:</u>

Multmat.asm

1. שימוש:
   a. ביצוע כפל מטריצות 4X4 מהזיכרון המדומה, ושמירת התוצאה בזיכרון המדומה.
2. מבנה:

a. פונקציית main הכוללת שתי לולאות (LOOP1_IN_MAIN, LOOP2_IN_MAIN).
b. פונקציית CREATE_CELL הנקראת בכל איטרציה, המחשבת את הערך בכל תא במטריצת התוצאה. מכילה את הלולאה LOOP_IN_CELL.
c. בסוף יש רצף שורות אתחול של זיכרון, לא כחלק מהקוד עצמו- אלא כדי ליצר תהליך שאינו הכפלת מטריצת 0 במטריצת 0.

3. פסאודו קוד:

```
Def main():
    For(int i=0; i<4;i++):
        For (int j=0;j<4;j++):
            Create_cell(i,j)
Def create_cell(int i, int j):
    Int tmp = 0
    For(int k=0;k<4;k++):
        Tmp = tmp + matA[i][k]*matB[k][j]
    matC[i][j] = tmp
```

a. $a0 -> i, $a1 -> j, $t0 -> k, $t1 -> tmp
b. אין הזזה בקוד הנ"ל של $sp, מאחר שלא בוצע שימוש ברגיסטרים שדורש זאת.

Binom.asm

1. שימוש:
   a. מחשב את התוצאה של $\binom{n}{k}$ עבור n, k חיוביים.
2. מבנה:
   a. טעינת ערכים מהזיכרון וכניסה ללולאה המרכזית.
   b. לולאה מרכזית - חלוקה למקרי קצה ולולאה כללית וקפיצה בהתאם.
      i. מקרי קצה – מגדילים את התוצאה ב-1.
      ii. מקרים כלליים – מאחסנים את כל הפרמטרים ואת כתובת החזרה ב-stack, וקוראים ללולאה הכללית פעמיים – פעם אחת עם n – 1, k – 1 ופעם אחת עם n – 1, k.
   c. שמירה מראש של ערכים לזכרון

Circle.asm

1. שימוש: יצירת מעגל בצבע לבן ברדיוס הנדרש.
2. מבנה:
   a. טעינת הרדיוס מהזיכרון ושמירת הרדיוס בריבוע
   b. לולאה חיצונית –
      i. איתחול קואורדינטת ה-X ל-0
      ii. חישוב המרחק הנוכחי בציר ה-Y בריבוע
   c. לולאה פנימית –
      i. חישוב המרחק הנוכחי בציר ה-X בריבוע
      ii. שמירת סכום המרחקים בריבוע (כל אחד בריבוע בנפרד)
      iii. בדיקה אל מול הרדיוס בריבוע

iv. צביעה בלבן במקרה והמרחק מהמרכז (128, 128) קטן או שווה לרדיוס

d. סוף הלולאה –

i. העלאת קואורדינטת ה-X כל פעם וקפיצה ללולאה הפנימית

ii. בכל 256 ריצות של הלולאה הפנימית, העלאת קואורדינטת ה-Y באחד וקפיצה ללולאה החיצונית.

Disktest.asm

1. שימוש:

a. העתקת תוכן הסקטורים 0-7 לסקטורים 1-8 בהתאמה. (לא "מחקנו" את התוכן של סקטור 0, מאחר שלא ראינו דרישה לכך- אלא רק דרישה להמצאות תוכן ב1-8).

2. מבנה:

a. פונקציית main הכוללת לולאה LOOP.

b. פונקציית MOVE_SECTOR הכוללת שתי לולאות (WHILE1, WHILE2) הדואגת לקרוא את תוכן סקטור מסוים, ולכתוב אותו לסקטור העוקב.

3. פסאודו קוד:

```
Def main():
        Diskbuffer = 0;
        For(int i=7; i>-1; i--):
                Move_sector(i);
Def move_sector(i):
        While (disk_not_ready){};
        Disksector = i;
        diskcmd = read;
        While (disk_not_ready){};
        Disksector = i+1;
        Diskcmd = write;
```

a. $a0 -> i

b. הבאפר נותר קבוע מאחר שבכל איטרציה הסקטור נקרא ואז נכתב, והתוכן נדרס באיטרציה הבאה (כי הוא כבר בדיסק).

c. אין הזזה בקוד הנ"ל של $sp, מאחר שלא בוצע שימוש ברגיסטרים שדורש זאת.