SIMP project documentation

Simulator:

- Firstly, init() function is executed, this includes initialization of all data holders (arrays, matrixes, etc.) within the C file scope, including nullifying IOregs, Monitor, PC counter, Error flag (internal for debugging), halt flag, not_in_ISR flag, irq2_next_stop flag, Processor registers.
  Then all input files that should be read from are opened and read entirely to appropriate global data holders.
  Only Irq2in.txt is read for one line only, which represents the first interruption point of irq2, later on, irq2in.txt is kept being read every time a new irq2_next_stop is reached, and the next one is loaded.

  Notes:
  hexStringToBinary() is used to parse a line from imemin.txt to 48 int array of bits.
  parseBitArray() is used to parse an 48 bit array to ParsedInstruction struct.
  A ParsedInstruction struct is used to store every instruction fields.

- Then "dynamic files" are opened in main(), that's because we want to write to them periodically and they will be passed as an argument to perform_instruction() which happens every cycle, later on.

- Then we reach a while that stops only on halt == 1 or Error == 1, which can occur only if there's an Error of IO operations or a halt was reached via the command 'halt'.

- Inside while:
  - Interrupt flag irq is set if irq_i_status and irq_i_enable are both 1 for the same 'i' for 0<=i<=2.
  - If an interrupt occurs, and not_in_ISR flag isn't raised, then PC is stored for later in irqreturn and then changed to ISR PC again of irqhandler.
  - Perform_instruction() is called (documentation in a moment)
  - IOprocedures() called()

- Perfrom_instruction() gets FILE* pointers to in order to pass them for writing periodically to output files: trace, HWtrace, leds, display.
  Need_to_increment_PC is set to 1 by default and may be changed for some instructions.
  Curr is ParsedInstruction struct which is loaded from a data holder of all instructions through parseBitArray with relevant PC.
  Immediates are checked for their unsigned value and if great enough that it means their msb is 1 and therefore sign-extended with 1's into Reg[1] or Reg[2] accordingly.

Write_to_trace() is called with PC, curr, and pointer to trace file as arguments in order to write one line to trace.txt.
Then curr.opcode is checked only one right opcode is implemented. For arithmetics, if rd is 0,1 or 2, nothing is being done.
For any kind of branch, need_to_increment_PC is set to 0.
IOregs read/write or memory read/write is done in this function's scope.
If IOreg is being written or read, a write_to_HWtrace is called.
If leds or display being written, a line being written to leds.txt or display.txt in this scope.
If opcode = 21 (halt), halt global flag being raised.
If need_to_increment_PC is still 1, PC is incremented.

- IOprocedures() is a function where many IOprocedures occur right after an instruction:
    - Timercurrent is evaluated against timermax and an interrupt is set if equal or timercurrent incremented if timerenable Is set.
    - Write to monitor data holder matrix is done with the help of monitordata and monitoraddr and that's if monitorcmd is set.
    - The disk is read or written if diskcmd is 1 or 2 and the disk is not occupied (diskstatus checked). Disksector and diskbuffer are used for writing or reading.
    - If Disk is occupied disktimer is incremented by 1 each cycle if lower then 1024.
    - If disktimer is 1024, irq1status is raised and disktimer is nullified.
    - If irq2_next_stop is equall to clks, raise irq2status and read next line from irq2in.txt to set the next irq2_next_stop.
    - Increment clks or nullify if was 0xFFFFFFFF.
    -
- After the while:

- Close all the files that have been used inside the while and are not needed anymore.

- Many "create_X" functions, each opens a txt file that is dependent only on the final states of the data holder like diskout.txt or regout.txt.
  After opening the file and writing it accordingly, the file is closed inside each "create_X".

- Return Error from main as an indicator if something went wrong (initially Error is 0).

Circle

The program initially sets the radius to memory address 0x100 via .word.
Then the program computes the radius squared. The program initializes two variables to represent the line number and column number, then it iterates through each pixel and computes its squared distance from the center via the formula:
(line_number – 128) ^ 2 + (column_number – 128) ^ 2.
If this number is less or equal to the radius squared, it jumps to color the pixel, by writing 255 to the address line_number * 256 + column_number as that's the appropriate address.
When done iterating the pixels and coloring the relevant, halt.

Disktest

The program waits for the disk to be ready (read diskstatus until it's 0) and then reads sector 7 to the start of the memory, checks every cycle for diskstatus again to change back to 0 and when it changes it writes a sector from the start of the memory to sector 8 and waits again for diskstatus to tell that the disk is ready for another operation again.
Now when prior sector 7 is copied to 8, it can be overwritten freely. Therefore do the same operation with sectors 6 and 7, then 5 and 6, etc. until doing this with 0 and 1.
Now iterate through 128 first lines of memory and delete them.
Now write to sector 0 from the start of the memory in order to nullify it because it's already moved to sector 1.
Halt.

אסמבלר

במימוש האסמבלר ראשית הגדרנו 3 משתנים מסוג struct, המשתנה הראשון הינו InsrtuctionMapping שבו קיימים השדות instr מסוג char ו opcode מסוג int , כל משתנה מגדיר הוראה שבו שמור שם ההוראה והמספר שלה.

המשתנה השני הינו RegisterMapping שבו קיימים השדות regname מסוג char ו regnumber מסוג int, כל משתנה מגדיר רגיסטר שבו שמור שם הרגיסטר והמספר שלו.

המשתנה השלישי הינו Labelmapping שבו קיימים השדות labname מסוג char ו lablinenum מסוג int, כל משתנה מגדיר תווית שקיימת בקוד, שבו שמור שם התווית ומספר ההוראה שבה היא מתחילה.

לאחר מכן הגדרנו מערך שנקרא instructions ובו הכנסנו את כל ההוראות הנתונות כמשתנים מסוג InsrtuctionMapping. בדומה לכך, הגדרנו מערך שנקרא registers ובו הכנסנו את כל הרגיסטרים הנתונים כמשתנים מסוג RegisterMapping. בנוסף, הגדרנו מערך שנקרא labels שבהמשך הקוד נכניס לתוכו את כל התוויות מהקוד כמשתנים מסוג Labelmapping.

בהמשך לכך, הגדרנו מס' פונקציות עזר, הפונקציה getOpcode מקבלת שם הוראה ומחזירה את מספרה ע"י חיפוש ההוראה במערך insgtruction.

באופן זהה, הפונקציה getRegisterNumber מקבלת את שם הרגיסטר ומחזיקה את מספרו ע"י חיפוש הרגיסטר במערך registers.

פונקציית עזר נוספת שהוגדרה היא convert_str_to_dec שתפקידה לסייע בהמרת מספר שכתוב כמחרוזת של תווים (chars) למספר מסוג int דצימלי. המספר שנקלט יכול להיות כתוב כמספר הקסה-דצימלי או כמספר דצימלי, טיפלנו בשני המקרים כך שהפונציה תחזיר עבורם את המספר כ int דצימלי.

כמו כן, הפונקציה getImmNumber מקבלת לתוכה את הערך של הimmediate, שיכול להיות שם של תווית או מספר שכתוב בפורמט דצימלי או הקסה-דצימלי. לכן ראשית הפונקציה בודקת אם התו הראשון בקלט הינו אות, אם כן אזי מדובר בתווית, נחפש את שם התווית במערך labels ונחזיר את מספר ההוראה שבה היא מתחילה. אחרת, נחזיר את מספר ה immediate כ int דצימלי בעזרת הפונקציה שהוגדרה קודם convert_str_to_dec.

לאחר מכן הגדרנו את הפונקציה assemblerInstruction שמקבלת לתוכה שורה מתוך קובץ הקלט שהוא קוד האסמבלי ואת קובץ הפלט. בראשית הפונקציה בדקנו אם מופיעה במהלך השורה הערה (comment) ואם כן, הגדרנו במיקומה את סוף השורה, כלומר התעלמנו ממנה. לאחר מכן, פירקנו את השורה למילים כשהתווים המגדירים את ההפרדה הם ',' ' ' '\t' ,'\n'. לאחר מכן שמרנו את המילים לפי הייצוג שלהם בפורמט קוד האסמבלי, כלומר הראשונה מייצגת הוראה, השנייה מייצגת את הרגיסטר rd וכך הלאה. בעזרת המילים ששמרנו ופונקצית העזר getRegisterNumber, getOpcode ו getImmNumber שמרנו בהתאמה את מספר ההוראה, מספרי הרגיסטרים ואת המספרים שמיוצגים בimmediate. כידוע, משתנה מסוג int מוגדר ע"י 32 ביטים בשפת C, לכן ע"מ להתאים את הערך הנשמר לגודל הimmediate שהוא 12 ביטים, ביצענו מאסקינג עם המספר 0xFFF (עבור מספר ההוראה ומספרי הרגיסטרים פעולה זו לא נחוצה). לבסוף, ביצענו הדפסה של שרשור הערכים הללו בקובץ הפלט, הרי שזוהי שורת הקוד בשפת המכונה.

המימוש הבא בקוד הוא הפונקציה word_Instruction, פונקציה זו מטפלת במקרה שההוראה המתבצעת בשורה היא הפסאודו-הוראה word. , הפונקציה מקבלת כקלט את השורה מקובץ הקלט, ופוינטרים ל2 מערכים – address ו value . השורה מופרדת למילים כשהתווים המגדירים את ההפרדה הם ' ' '\t' ,'\n'. לאחר מכן שמרנו את המילה השניה שמגדירה את הכתובת במערך address ואת המילה השלישית שמגדירה את הערך במערך value.

הפונקציה האחרונה שממומשת בקוד היא הפונקציה main. בתחילת הפונקציה ביצענו פתיחה של הקבצים, קובץ אחד הוא קובץ הקלט, האסמבלי, שממנו מבוצעת קריאה, ו2 קבצים נוספים הם קבצי הפלט imemin.txt ו dmemin.txt שאליהם מבוצעת כתיבה. בנוסף, איתחלנו מערך בגודל 4096 להיות מערך של אפסים שנקרא memory.

לאחר מכן הגדרנו 3 מספרים מסוג int , numofline אשר מייצג ספירה של מספר השורות שהן שורות קוד בקובץ הקלט. numoflabels שמייצג ספירה של מספר התוויות בקוד. ו numofcharinline שמייצג ספירה של התווים בשורה.

הצעד הבא בפונקציה הוא לעבור על כל קובץ הקלט ולהוציא ממנו את כל התוויות הקיימות ולשמור אותן במערך labels. בעזרת לולאת while, אנו עוברים על כל השורות בקובץ, תוך התעלמות מ"white spaces", אם אנו נתקלים בשורה ריקה או שורה שמכילה רק הערה, נדלג עליה. אחרת, נבדוק האם המילה הראשונה בשורה מכילה את התו ':', משום שתחילת תווית מוגדרת ע"י שמה ואחריו ':'. אם כן, הגענו לתווית!, נמחק את התו ':' שצמוד לשם ונשמור את התווית כמשתנה במערך labels מסוג Labelmapping שבו שמורים שם התווית ומספר שורת הקוד בה התווית מתחילה (בעזרת numofline). לבסוף, אם באותה השורה, אחרי התווית, לא מתחילה הוראה או שמתחילה פסאודו-הוראה, נמשיך לשורה הבאה. אחרת, נגדיל את ספירת מספר שורות הקוד.

לאחר מכן, נבצע חזרה של הפוינטר לתחילת קובץ הקלט, ונבצע טיפול עבור כל שורת הוראה בו. בעזרת לולאת while, אנו עוברים על כל השורות בקובץ, תוך התעלמות מ"white spaces", אם אנו נתקלים בשורה ריקה או שורה שמכילה רק הערה, נדלג עליה. לאחר מכן נבדוק מהי המילה הראשונה בשורה, אם היא 'word.' - נקרא לפונקציה word_Instruction ואחריה נמיר את הכתובת והערך שהוחזרו ממנה לint דצימלי בעזרת הפונקציה convert_str_to_dec, ובעזרת אלו נגדיר במערך הmemory במיקום הכתובת את הערך המתאים. ונמשיך לשורה הבאה.

אם המילה הראשונה בשורה היא תווית, נתעלם מ"white spaces" אחריה ונבדוק אם מתחילה הוראה באותה השורה, אם כן, נקרא לפונקציה assembleInstruction עם מיקום תחילת ההוראה בשורה. אחרת, נמשיך לשורה הבאה.

בכל מקרה אחר, כלומר המילה הראשונה אינה תווית או פסאודו-הוראה, נקרא לפונקציה assembleInstruction ע"מ להמיר את שורת הקוד לשפת מכונה בקובץ הפלט imemin.txt.

לבסוף, נמלא את שורות קובץ הפלט dmemin.txt כך שכל איבר במערך memory יכנס לשורה בקובץ בפורמט הקסה-דצימלי. ונדאג לסגור את הקבצים שפתחנו.

טסט 1 – Mulmat

הבהרה - הכוונה ב"מספר איבר במטריצה" היא: (כל מספר במטריצת ההמחשה הבאה מייצג את מספר האיבר ונמצא במיקום שלו)

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix}$$

בתחילת קוד האסמבלי ביצענו כתיבה לזיכרון של ערכי איברי 2 מטריצות במיקומי הזיכרון המתאימים. המטריצות שאותם איתחלנו הינן (מטריצה 1 משמאל ומטריצה 2 מימין):

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ -1 & -2 & -3 & -4 \\ -4 & -3 & -2 & -1 \\ 4 & 3 & 2 & 1 \end{pmatrix} \quad \begin{pmatrix} 2 & 2 & 2 & 2 \\ 0 & 1 & 0 & 1 \\ -4 & 3 & -2 & 1 \\ 0 & 5 & 6 & 0 \end{pmatrix}$$

לאחר מכן איתחלנו את $s0 ל-0 כאשר הוא ייצג את מספר השורה במטריצה 1, ואת $t2 ל-0 כאשר הוא ייצג לנו את מספר האיבר שאותו נכניס למטריצת התוצאה.

<u>תווית 'L1'-</u>

איתחול $a2 ל-0 שמייצג את מספר העמודה במטריצה 2.

<u>תווית 'L3'-</u>

איתחול של $s2 ל-0 שמייצג לנו את סכום המכפלות של איברי שורה ממטריצה 1 ועמודה ממטריצה 2 (כפי שמוגדרת מכפלת מטריצות)

ואיתחול של $s1 ל-0 שמייצג את מספר העמודה במטריצה 1.

<u>תווית 'L2'-</u>

נגדיר את $t0 להיות מספר השורה במטריצה 1 כפול 4 ועוד מספר העמודה במטריצה 1. כלומר $t0 מייצג את מספר האיבר במטריצה 1 שאותו נכפול.

לאחר מכן נטעין את ערך האיבר הזה מהזיכרון ע"י כך שמיקומו בזיכרון הוא מיקום האיבר הראשון של מטריצה 1 (0x100) ועוד מספר האיבר במטריצה, ונכניס את הערך ל$a0.

באופן דומה נגדיר את $t1 להיות מספר העמודה במטריצה 1 כפול 4 ועוד מספר העמודה במטריצה 2. כלומר $t1 מייצג את מספר האיבר במטריצה 2 שאותו נכפול.

לאחר מכן נטעין את ערך האיבר הזה מהזיכרון ע"י כך שמיקומו בזיכרון הוא מיקום האיבר הראשון של מטריצה 2 (0x110) ועוד מספר האיבר במטריצה, ונכניס את הערך ל$a1.

בשורה הבאה, נבצע את מכפלת האיברים הנוכחיים משתי המטריצות ונוסיף את ערך זה לערך השמור ב$s2.

נגדיל את $s1 ב1 כלומר נקדם את מספר העמודה במטריצה 1.

אם מספר העמודה במטריצה 1 שונה מ4, כלומר אם עדיין לא עברנו על כל העמודות בשורה הנוכחית, נקפוץ לתווית L2.

לאחר שעברנו על כל העמודות של מטריצה 1 בשורה הנוכחית, נשמור את הערך שנסכם ב$s2 כאיבר במטריצת התוצאה, נעשה זאת ע"י איחסון בזיכרון במיקום שהוא מיקום האיבר הראשון של מטריצת התוצאה (0x120) ועוד מספר האיבר במטריצה זו ששמור כזכור ב $t2.

לאחר מכן נקדם את מספר האיבר במטריצת התוצאה ($t2) ב-1.

בנוסף, נקדם את מספר העמודה במטריצה 2, ששמור ב $a2, ב-1.

אם מספר העמודה במטריצה 2 שונה מ4, כלומר אם עדיין לא עברנו על כל העמודות נקפוץ לתווית L3, עד שבעצם נמלא שורה של ערכים במטריצת התוצאה.

נגדיל את $s0 ב1 כלומר נקדם את מספר השורה במטריצה 1.

אם מספר השורה במטריצה 1 שונה מ4, כלומר אם עדיין לא עברנו על כל השורות נקפוץ לתווית L2, עד שבעצם נמלא את כל מטריצת התוצאה.

קיבלנו את הנדרש, לכן נסיים את הריצה ע"י פקודת halt.

טסט 2 – binom

בתחילת קוד האסמבלי ביצענו כתיבה לזיכרון של ערכי n ו k במיקומי הזיכרון המתאימים. הערכים אותם איתחלנו הינם n=5 k=2.

ראשית, הגדרנו פוינטר למחסנית ע"י שימוש ב $sp במיקום 2048.

לאחר מכן, שמרנו את ערך n ע"י קריאה ממיקומו בזיכרון ב$a0

ושמרנו את ערך k ע"י קריאה ממיקומו בזיכרון ב$a1

ביצענו קפיצה לתווית binom שתחשב לנו את הערך binom(n,k) רקורסיבית כאשר נשמר הpc של ההוראה הבאה ב$ra .

לאחר מכן הערך שהוחזר לנו מהקפיצה הוא הערך שנדרשנו לחשב, ונאחסן אותו במיקומו בזיכרון (0x102).

קיבלנו את הנדרש, לכן נסיים את הריצה ע"י פקודת halt.

תווית 'binom' -

ראשית נגדיר מקום במחסנית ל4 איברים ע"י הקטנת $sp ב-4.

נשמור במחסנית את $s0, $ra, $a0,$a1 הנוכחיים.

נגדיר תנאי התחלה, אם n=k או ש k=0, נקפוץ לתווית L3.

אם הקפיצה לא בוצעה, כלומר אנו לא עומדים בתנאי ההתחלה של הקריאה הרקורסיבית, נקפוץ לתווית L1.

תווית 'L3' –

כפי שהגדרנו, אם הגענו לתווית זו אזי אנו עומדים בתנאי ההתחלה ולכן נגדיר את $v0 שזהו ערך ההחזרה, להיות 1.

לאחר מכן, נקפוץ לתווית L2 ע"מ לסיים את הקריאות הרקורסיביות.

תווית 'L1' –

בתווית זו אנו מחשבים את ערך הקריאה הרקורסיבית, תחילה, נחשב את n-1 ונשמור אותו ב$a0.

כעת, נבצע קריאה רקורסיבית, כלומר נקפוץ לתווית binom שממנה נקבל את הערך המוחזר binom(n-1,k) כאשר נשמר הpc של ההוראה הבאה ב$ra .

נשמור את הערך המוחזר של binom(n-1,k) ב$s0.

נחשב את k-1 ונשמור אותו ב$a1.

כעת, נבצע קריאה רקורסיבית, כלומר נקפוץ לתווית binom שממנה נקבל את הערך המוחזר binom(n-1,k-1) כאשר נשמר הpc של ההוראה הבאה ב$ra .

נסכום את ערכי 2 ההחזרות מהקריאות הרקורסיביות שביצענו, כלומר נבצע את החישוב binom(n-1,k)+binom(n-1,k-1) ונשמור אותו ב$v0.

כעת נשחזר את הערכים של הקריאה הנוכחית ששמרנו במחסנית, כלומר את $s0, $ra, $a0, $a1.

תווית 'L2' –

נשחרר את ארבעת המקומות בזיכרון ע"י הגדלת $sp ב-4.

ולבסוף, נבצע חזרה לכתובת השמורה ב$ra (כלומר נחזור למי שקרא לפונ' הרקורסיבית).