אוניברסיטת תל-אביב, הפקולטה להנדסה

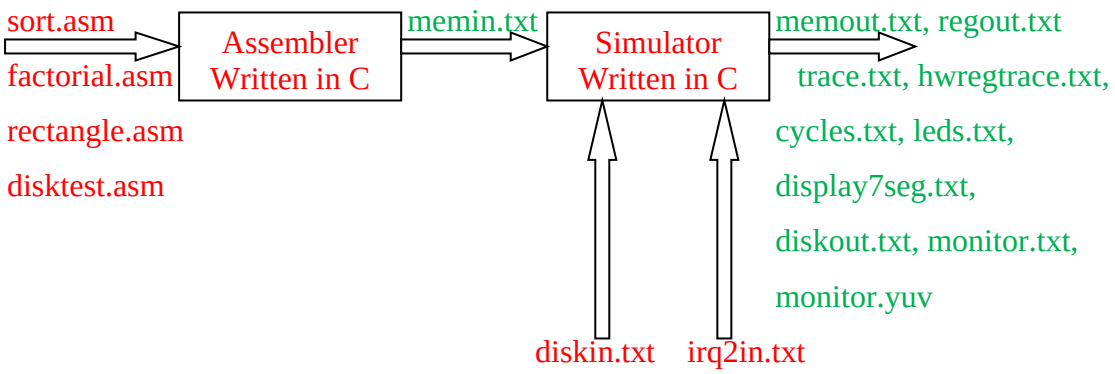
פרויקט בקורס: מבנה המחשב 0512.4400

שנת הלימודים תשפ"ה, סמסטר ב'

בפרויקט נתרגל את נושאי שפת המחשב, קלט/פלט, וכמו כן נתרגל את יכולות התכנות שלנו בשפת סי. נממש אסמבלר וסימולטור (תוכניות נפרדות), ונכתוב תוכניות בשפת אסמבלי עבור מעבד RISC בשם SIMP, אשר דומה למעבד MIPS אבל פשוט ממנו.

הסימולטור יסמלץ את מעבד ה- SIMP, וכמו כן מספר התקני קלט/פלט: נורות, תצוגת 7-segment, מוניטור מונוכרומטי ברזולוציה 256x256, ודיסק קשיח.

הדיאגרמה הבאה ממחישה את הפרויקט:



החלקים שאותם תכתבו בפרויקט ידנית מסומנים בצבע אדום, ואילו קבצי פלט שייוצרו אוטומטית ע"י תוכנות האסמבלר והסימולטור שתכתבו מסומנים בצבע ירוק.

רגיסטרים

מעבד SIMP מכיל 16 רגיסטרים, שכל אחד מהם 32 ברוחב ביטים. שמות הרגיסטרים, מספרם, ותפקיד כל אחד מהם בהתאם ל- calling conventions, נתונים בטבלה הבאה:

Register Number	Register Name	Purpose
0	$zero	Constant zero
1	$imm	Sign extended imm
2	$v0	Result value
3	$a0	Argument register

4	$a1	Argument register
5	$a2	Argument register
6	$a3	Argument register
7	$t0	Temporary register
8	$t1	Temporary register
9	$t2	Temporary register
10	$s0	Saved register
11	$s1	Saved register
12	$s2	Saved register
13	$gp	Global pointer (static data)
14	$sp	Stack pointer
15	$ra	Return address

שמות הרגיסטרים ותפקידם דומים למה שראינו בהרצאה ובתירגולים עבור מעבד MIPS, בהבדלים הבאים: הרגיסטרים $zero, $imm הינם רגיסטרים מיוחדים שלא ניתן לכתוב אליהם, ויכולים להיות בשימוש בכל מקום שבו יש שימוש ב- source operand:

רגיסטר $zero הינו זהותית אפס.

רגיסטר $imm אינו רגיסטר אמיתי (לא מכיל פליפלופים) אלא מכיל את הקבוע כפי שקודד בהוראת האסמבלי. יש שתי אפשרויות לקודד את הקבוע בהתאם לגודלו, כפי שמתואר בדף הבא.

הוראות אשר כותבות ל- $zero, $imm לא משנות את ערכם.

זיכרון ראשי, סט ההוראות וקידודם

הזיכרון הראשי הינו ברוחב 32 סיביות ובעומק 4096 שורות. הוראה במעבד SIMP מקודדת בשורה אחת או בשתי שורות בזיכרון, כאשר השורה הראשונה נתונה בפורמט הבא:

First Row						
31:24	23:20	19:16	15:12	11:9	8	7:0
opcode	rd	rs	rt	reserved (set to 0)	bigimm	imm8

הביט bigimm קובע האם יש שימוש בשורה נוספת בקידוד ההוראה: במידה וערכו 1, ההוראה תשתמש בשורה נוספת שתכיל קבוע ברוחב 32 ביטים.

Second Row 31:0 (only used if bigimm == 1)
imm32

רגיסטר ה- PC הינו ברוחב 12 ביטים. כדי להתקדם להוראה הבאה (במידה ואין קפיצה), PC מתקדם באחד עבור ההוראות אשר מקודדות בשורה אחת, או בשניים עבור ההוראות אשר מקודדות בשתי שורות.

בזמן פענוח ההוראה, הקבוע ("רגיסטר 1") נקבע באחד משני אופנים:

במידה ו- bigimm == 0, הקבוע יהיה imm8 לאחר הרחבת סימן (שכפול ביט 7 של הקבוע לביטים 31:8).

במידה ו- bigimm == 1, הקבוע נלקח מ- 32 הביטים בתא השני בקידוד ההוראה, imm32, ו- imm8 אינו בשימוש.

הוראה אשר מקודדת בשורה אחת בזיכרון (bigimm == 0) מתבצעת במחזור שעון אחד.

הוראה אשר מקודדת בשני תאים בזיכרון (bigimm == 1) מתבצעת בשני מחזורי שעון.

האופקודים הנתמכים ע״י המעבד ומשמעות כל הוראה נתונים בטבלה הבאה:

Opcode Number	Name	Meaning
0	add	R[rd] = R[rs] + R[rt]
1	sub	R[rd] = R[rs] – R[rt]
2	mul	R[rd] = R[rs] * R[rt]
3	and	R[rd] = R[rs] & R[rt]
4	or	R[rd] = R[rs] | R[rt]
5	xor	R[rd] = R[rs] ^ R[rt]
6	sll	R[rd] = R[rs] << R[rt]
7	sra	R[rd] = R[rs] >> R[rt], arithmetic shift with sign extension
8	srl	R[rd] = R[rs] >> R[rt], logical shift
9	beq	if (R[rs] == R[rt]) pc = R[rd]
10	bne	if (R[rs] != R[rt]) pc = R[rd]
11	blt	if (R[rs] < R[rt]) pc = R[rd]
12	bgt	if (R[rs] > R[rt]) pc = R[rd]
13	ble	if (R[rs] <= R[rt]) pc = R[rd]
14	bge	if (R[rs] >= R[rt]) pc = R[rd]
15	jal	R[rd] = next instruction address, pc = R[rs]
16	lw	R[rd] = MEM[R[rs]+R[rt]]
17	sw	MEM[R[rs]+R[rt]] = R[rd]
18	reti	PC = IORegister[7]
19	in	R[rd] = IORegister[R[rs] + R[rt]]
20	out	IORegister [R[rs]+R[rt]] = R[rd]
21	halt	Halt execution, exit simulator

קלט/פלט

המעבד תומך בקלט/פלט באמצעות הוראות in ו- out הניגשות למערך "רגיסטרי חומרה" כמפורט בטבלה מטה. הערכים ההתחלתיים של רגיסטרי החומרה ביציאה מריסט הם 0.

IORegister Number	Name	number bits	Meaning
0	irq0enable	1	IRQ 0 enabled if set to 1, otherwise disabled.
1	irq1enable	1	IRQ 1 enabled if set to 1, otherwise disabled.
2	irq2enable	1	IRQ 2 enabled if set to 1, otherwise disabled.
3	irq0status	1	IRQ 0 status. Set to 1 when irq 0 is triggered.
4	irq1status	1	IRQ 1 status. Set to 1 when irq 1 is triggered.
5	irq2status	1	IRQ 2 status. Set to 1 when irq 2 is triggered.
6	irqhandler	12	PC of interrupt handler
7	irqreturn	12	PC of interrupt return address
8	clks	32	cyclic clock counter. Starts from 0 and increments every clock. After reaching 0xffffffff, the counter rolls back to 0.
9	leds	32	Connected to 32 output pins driving 32 leds. Led number i is on when leds[i] == 1, otherwise its off.
10	display7seg	32	Connected to 7-segment display of 8 letters. Each 4 bits displays one digit from 0 – F, where bits 3:0 control the rightmost digit, and bits 31:28 the leftmost digit.
11	timerenable	1	1: timer enabled 0: timer disabled
12	timercurrent	32	current timer counter
13	timermax	32	max timer value
14	diskcmd	2	0 = no command 1 = read sector 2 = write sector
15	disksector	7	sector number, starting from 0.

16	diskbuffer	12	Memory address of a buffer containing the sector being read or written. Each sector will be read/written using DMA in 128 words.
17	diskstatus	1	0 = free to receive new command 1 = busy handling a read/write commad
18-19	reserved		Reserved for future use
20	monitoraddr	16	Pixel address in frame buffer
21	monitordata	8	Pixel luminance (gray) value (0 – 255)
22	monitorcmd	1	0 = no command 1 = write pixel to monitor

פסיקות

מעבד SIMP תומך ב- 3 פסיקות: irq0, irq1, irq2.
פסיקה 0 משויכת לטיימר, וקוד האסמבלי יכול לתכנת כל כמה זמן הפסיקה תתרחש.

פסיקה 1 משויכת לדיסק הקשיח המסומלץ, באמצעותה הדיסק מודיע למעבד כאשר סיים לבצע הוראת קריאה או כתיבה.

פסיקה 2 מחוברת לקו חיצוני למעבד irq2. קובץ קלט לסימולטור קובע מתי הפסיקה מתרחשת.

במחזור השעון בו הפסיקה מתקבלת, מדליקים את אחד הרגיסטרים irq0status, irq1status, irq2status בהתאמה. אם מספר פסיקות מתקבלות באותו מחזור שעון, ידלקו בהתאמה מספר רגיסטרי סטטוס.

בכל מחזור שעון המעבד בודק את הסיגנל:

irq = (irq0enable & irq0status) | (irq1enable & irq1status) | (irq2enable & irq2status)

במידה ו- irq == 1, ואנחנו לא במחזור השני של טיפול בהוראה שלוקחת שני מחזורי שעון (bigimm == 1), והמעבד לא נמצא כרגע בתוך שגרת הטיפול בפסיקה, המעבד קופץ לשגרת הטיפול בפסיקה שכתובתה בזיכרון נתונה ברגיסטר חומרה irqhandler. כלומר במחזור שעון זה מתבצעת ההוראה ב- PC = irqhandler במקום ב- PC המקורי. באותו מחזור שעון ה- PC המקורי נשמר לתוך רגיסטר חומרה irqreturn.

לעומת זאת במידה ו- irq == 1 והמעבד עדיין נמצא בתוך שגרת הטיפול בפסיקה קודמת (כלומר עדיין לא הריץ את הוראת ה- reti), המעבד יתעלם, לא יקפוץ וימשיך להריץ את הקוד כרגיל בתוך שגרת הפסיקה (כאשר המעבד יחזור מהפסיקה, הוא יבדוק שוב את irq ואם יהיה צורך יקפוץ שוב לשגרת הפסיקה).

קוד האסמבלי של שגרת הפסיקה יבדוק את הביטים של irqstatus, ולאחר טיפול מתאים בפסיקה יכבה את הביטים.

חזרה משגרת הפסיקה מתבצעת באמצעות הוראת reti, שתציב PC = irqreturn.

טיימר

מעבד SIMP תומר בטיימר של 32 ביטים, המחובר לפסיקה irq0. הוא מאופשר כאשר timerenable = 1.

ערך מונה הטיימר הנוכחי שמור ברגיסטר חומרה timercurrent. בכל מחזור שעון שבו הטיימר מאופשר, רגיסטר timercurrent מקודם באחד.

במחזור השעון שבו timercurrent = timermax, מדליקים את irqstatus0. במחזור שעון זה במקום לקדם את timercurrent, מאפסים אותו חזרה לאפס.

נורות לד

למעבד SIMP מחוברים 32 נורות. קוד האסמבלי מדליק/מכבה נורות ע״י כתיבה של מילה ברוחב 32 ביטים לרגיסטר החומרה leds, כאשר ביט 0 מדליק/מכבה את נורה 0 (הימנית), וביט 31 את נורה 31 (השמאלית).

מוניטור

למעבד SIMP מחובר מוניטור מונוכרומטי ברזולוציה 256x256 פיקסלים. כל פיקסל מיוצג ע״י 8 ביטים שמייצגים את גוון האפור של הפיקסל (luminance) כאשר 0 מסמן צבע שחור, 255 צבע לבן, וכל מספר אחר בתחום מתאר גוון אפור בין שחור ללבן באופן לינארי.

במסך יש frame buffer פנימי בגודל 256x256 המכיל את ערכי הפיסקלים שכעת מוצגים על המסך. בתחילת העבודה כל הערכים מכילים אפס. הבפר מכיל שורות של 256 בתים שמתאימים לסריקת המסך מלמעלה למטה. כלומר שורה 0 בבפר מכילה את הפיסקלים של השורה העליונה במסך. בכל שורה סריקת הפיקסלים במסך הינה משמאל לימין.

רגיסטר monitoraddr מכיל את האופסט בבפר של הפיקסל שאותו המעבד רוצה לכתוב.
רגיסטר monitordata מכיל ערך הפיקסל שאותו המעבד רוצה לכתוב.
רגיסטר monitorcmd משמש עבור כתיבה של פיקסל. במחזור השעון שבו יש כתיבה monitorcmd=1 באמצעות הוראת out, מתבצע עדכון של הפיסקל שתוכנו ברגיסטר monitordata על המסך.
קריאה מרגיסטר monitorcmd באמצעות הוראת in תחזיר את הערך 0.

דיסק קשיח

למעבד SIMP מחובר דיסק קשיח המורכב מ- 128 סקטורים, כאשר כל סקטור מכיל 128 שורות ברוחב 32 ביטים. הדיסק מחובר לפסיקה מספר 1, irq1, ומשתמש ב- DMA להעתקת הסקטור מהזיכרון לדיסק או להיפך.

תוכנו ההתחלתי של הדיסק הקשיח נתון בקובץ הקלט diskin.txt, ותוכן הדיסק בסיום הריצה ייכתב לקובץ diskout.txt.

לפני מתן הוראת קריאה או כתיבה של סקטור לדיסק הקשיח, קוד האסמבלי בודק שהדיסק פנוי לקבלת הוראה חדשה ע״י בדיקת רגיסטר חומרה diskstatus.

במידה והדיסק פנוי, כותבים לרגיסטר disksector את מספר הסקטור שרוצים לקרוא או לכתוב, ולרגיסטר diskbuffer את הכתובת בזיכרון. רק לאחר ששני רגיסטרים אלו מאותחלים, נותנים הוראת כתיבה או קריאה ע״י כתיבה לרגיסטר חומרה diskcmd.

זמן הטיפול של הדיסק בהוראת קריאה או כתיבה הוא 1024 מחזורי שעון. במהלך זמן זה יש להעתיק את תוכן הבפר לדיסק במידה והייתה כתיבה, או להיפך להעתיק את תוכן הסקטור לבפר אם הייתה קריאה.

כל עוד לא עברו 1024 מחזורי שעון מקבלת ההוראה, רגיסטר diskstatus יסמן שהדיסק עסוק.

לאחר 1024 מחזורי שעון, במקביל ישונו רגיסטר ה- diskcmd ו- diskstatus לערך 0, והדיסק יודיע על פסיקה ע״י הדלקת irqstatus1.

הסימולטור

הסימולטור מסמלץ את לולאת ה- fetch-decode-execute. בתחילת הריצה PC=0. בכל איטרצייה מביאים את ההוראה הבאה בכתובת ה- PC, מפענחים את ההוראה בהתאם לקידוד, ואח״כ מבצעים את ההוראה. בסיום ההוראה מעדכנים את PC לערך PC+1 או PC+2 (כתלות בהאם ההוראה מקודדת בתא בודד או בשני תאים), אלא אם כן בצענו הוראת קפיצה שמעדכנת את ה- PC לערך אחר. סיום הריצה ויציאה מהסימולטור מתבצע כאשר מבצעים את הוראת ה- HALT.

הסימולטור יכתב בשפת סי ויקומפל לתוך command line application אשר מקבל 13 command line parameters לפי שורת ההרצה הבאה:

sim.exe memin.txt diskin.txt irq2in.txt memout.txt regout.txt trace.txt hwregtrace.txt cycles.txt leds.txt display7seg.txt diskout.txt monitor.txt monitor.yuv

הקובץ **memin.txt** הינו קובץ קלט בפורמט טקסט אשר מכיל את תוכן הזיכרון הראשי בתחילת הריצה. כל שורה בקובץ מכילה תוכן שורה בזיכרון, החל מכתובת אפס, בפורמט של 8 ספרות הקסאדצימליות. במידה ומספר השורות בקובץ קטן מ- 4096, ההנחה הינה ששאר הזיכרון מעל הכתובת האחרונה שאותחלה בקובץ, מאופס. ניתן להניח שקובץ הקלט תקין.

הקובץ **diskin.txt** הינו קובץ קלט, שמכיל את תוכן הדיסק הקשיח בתחילת הריצה, כאשר כל שורה מכילה 8 ספרות הקסאדצימליות. במידה ומספר השורות בקובץ קטן מגודל הדיסק, ההנחה הינה ששאר הדיסק מעל הכתובת האחרונה שאותחלה בקובץ, מאופס.

הקובץ **irq2in.txt** הינו קובץ קלט, המכיל את מספרי מחזורי השעון שבהם קו הפסיקה החיצוני irq2 עלה ל- 1, כל מחזור שעון כזה בשורה נפרדת בסדר עולה. הקו כל פעם עולה ל- 1 למחזור שעון בודד ואז יורד חזרה לאפס (אלא אם כן מופיעה שורה נוספת בקובץ עבור מחזור השעון הבא).

שלושת קבצי הקלט צריכים להיות קיימים אפילו אם בקוד שלכם אין בהם שימוש (לדוגמא גם עבור קוד אסמבלי שאינו משתמש בדיסק הקשיח יהיה קיים קובץ קלט diskin.txt, כאשר מותר גם להשאיר את תוכנו ריק).

הקובץ **memout.txt** הינו קובץ פלט, באותו פורמט כמו memin.txt, שמכיל את תוכן הזיכרון הראשי בסיום הריצה.

הקובץ **regout.txt** הינו קובץ פלט, שמכיל את תוכן הרגיסטרים R2-R15 בסיום הריצה (שימו לב שאין להדפיס את הקבועים R0 – R1). כל שורה תיכתב ב- 8 ספרות הקסאדצימליות.

הקובץ **trace.txt** הינו קובץ פלט, המכיל שורת טקסט עבור כל הוראה שבוצעה ע״י המעבד בפורמט הבא:

CYCLE PC INST R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15

כל שדה מודפס בספרות הקסאדצימליות.

CYCLE הינו מחזור השעון שבו מתבצעת ההוראה ומקודד ב- 8 ספרות. במידה והוראה מתבצעת בשני מחזורי שעון (bigimm == 1), בשדה זה יודפס מחזור השעון השני מבין השניים.

ה- PC הינו ה- Program Counter של ההוראה (מודפס ב- 3 ספרות), ה- INST הינו קידוד התא הראשון של ההוראה כפי שנקרא מהזיכרון (8 ספרות), ואח"כ יש את תוכן הרגיסטרים **לפני** ביצוע ההוראה (כלומר את תוצאת הביצוע ניתן לראות רק ברגיסטרים של השורה הבאה). כל רגיסטר מודפס ב- 8 ספרות.

בשדה R0 יש לכתוב 8 אפסים. בשדה R1 יש לכתוב את תוכן הקבוע ב- 32 סיביות לאחר בצוע הרחבת סימן במידה ונדרש.

הקובץ **hwregtrace.txt** הינו קובץ פלט, המכיל שורת טקסט עבור כל קריאה או כתיבה לרגיסטר חומרה (באמצעות הוראות in ו- out) בפורמט הבא:

CYCLE READ/WRITE NAME DATA

כאשר השדה CYCLE הוא מספר מחזור השעון ב- 8 ספרות הקסאדצימאליות.

השדה הבא מכיל READ או WRITE בהתאם להאם קוראים או כותבים לרגיסטר החומרה.

השדה NAME מכיל את שם רגיסטר החומרה כפי שמופיע בטבלה.

השדה DATA מכיל את הערך שנכתב או נקרא ב- 8 ספרות הקסאדצימליות.

הקובץ **cycles.txt** הינו קובץ פלט, שמכיל את מספר מחזורי השעון שרצה התוכנית, ב- 8 ספרות הקסאדצימליות.

הקובץ **leds.txt** מכיל את סטטוס 32 הנורות. בכל מחזור שעון שאחת הנורות משתנה (נדלקת או נכבית), כותבים שורה עם שני מספרים ורווח ביניהם: המספר השמאלי הינו מחזור השעון, ב- 8 ספרות הקסא, והמספר הימני מצב כל 32 הנורות ב- 8 ספרות הקסאדצימליות.

הקובץ **display7seg.txt** מכיל את תצוגת ה- 7-segment display. בכל מחזור שעון שהתצוגה משתנה, כותבים שורה עם שני מספרים ורווח ביניהם: המספר השמאלי הינו מחזור השעון, ב- 8 ספרות הקסא, והמספר הימני הינו התצוגה, ב- 8 ספרות הקסאדצימליות.

הקובץ **diskout.txt** הינו קובץ פלט, באותו פורמט כמו diskin.txt, שמכיל את תוכן הדיסק הקשיח בסיום הריצה.

הקובץ **monitor.txt** מכיל את ערכי הפיסקלים שבמסך בסיום הריצה. כל שורה מכילה ערך פיסקל בודד (8 ביטים) בשתי ספרות הקסאדצימליות, כאשר סריקת המסך היא מלמעלה למטה, ומשמאל לימין. לדוגמא השורה הראשונה בקובץ מכילה את ערך הפיקסל בצד שמאל למעלה. במידה ומספר השורות בקובץ קטן ממספר הפיקסלים במסך, ההנחה היא ששאר הפיקסלים מכילים אפס.

הקובץ **monitor.yuv** הינו קובץ בינארי אשר מכיל את אותו דאטא כמו monitor.txt, וניתן להציגו על המסך באמצעות התוכנה yuvplayer:

https://github.com/Tee0125/yuvplayer

כאשר בפרמטרים בוחרים size = 256x256 ו- color = Y.

האסמבלר

כדי שיהיה נוח לתכנת את המעבד וליצור את תמונת הזיכרון בקובץ memin.txt, נכתוב בפרויקט גם את תוכנית האסמבלר. האסמבלר יכתב בשפת סי, ויתרגם את תוכנית האסמבלי שכתובה בטקסט בשפת אסמבלי, לשפת המכונה. ניתן להניח שקובץ הקלט תקין.

בדומה לסימולטור, האסמבלר הינו command line application עם שורת ההרצה הבאה:

asm.exe program.asm memin.txt

קובץ הקלט program.asm מכיל את תוכנית האסמבלי, קובץ הפלט memin.txt מכיל את תמונת הזיכרון הראשי, ומשמש אח"כ כקובץ קלט לסימולטור.

כל שורת קוד בקובץ האסמבלי מכילה 5 פרמטרים כמפורט מטה, כאשר הפרמטר הראשון הינו האופקוד, והפרמטרים מופרדים ע"י סימני פסיק. לאחר הפרמטר האחרון מותר להוסיף את הסימן # והערה מצד ימין, לדוגמא:

```
# opcode, rd, rs, rt, imm
add $t3, $t2, $t1, 0                # $t3 = $t2 + $t1
add $t1, $t1, $imm, 2               # $t1 = $t1 + 2
add $t1, $imm, $imm, 2              # $t1 = 2 + 2 = 4
```

בכל הוראה, יש שלוש אפשרויות עבור שדה הקבוע imm:

- ניתן לשים שם מספר דצימלי, חיובי או שלילי.
- ניתן לשים מספר הקסאדצימלי שמתחיל ב- 0x ואז ספרות הקסאדצימליות. במידה ויש פחות מ- 8 ספרות הקסא, ההבנה היא כאילו הספרות משמאל מכילות אפסים.
- ניתן לשים שם סימבולי (שמתחיל באות). במקרה זה הכוונה ל- label, כאשר label מוגדר בקוד ע"י אותו השם ותוספת נקודותיים.

כדי שיהיה נוח לכתוב את הקוד, בקוד האסמבלי אין הגדרה ספציפית לשדה bigimm אלא האסמבלר כאשר הוא מקודד את ההוראה לשפת המכונה, קובע את bigimm באופן הבא:

במידה והקבוע הינו label סימבולי, לשם פשטות מימוש המעבר השני, ההוראה תקודד בשני תאים בזיכרון עם bigimm = 1 (אפילו אם בסופו של דבר כתובת הלייבל תהיה מספר קטן).

אחרת, האסמבלר יבחן האם הקבוע נכנס בתחום 128- עד 127+:

במידה ואפשר, ההוראה תקודד בתא בודד בזיכרון עם bigimm = 0.

במידה ואי אפשר (כלומר הקבוע לא נכנס ב- 8 ביטים בפורמט המשלים ל- 2), ההוראה תקודד בשני תאים בזיכרון עם bigimm = 1.

דוגמאות:

```
        bne $imm, $t0, $t1, L1          # if ($t0 != $t1) goto L1
                                        # (reg1 = address of L1)
        beq $imm, $zero, $zero, L2      # jump to L2 (reg1 = address L2)
L1:
        sub $t2, $t2, $imm, 1           # $t2 = $t2 – 1 (reg1 = 1)
L2:     add $t1, $zero, $imm, L3        # $t1 = address of L3
        beq $t1, $zero, $zero,  0       # jump to the address specified in reg $t1
L3:     jal $ra, $imm, $zero, L4        # function call L4, save return addr in $ra
        halt $zero, $zero, $zero,  0    # halt execution
```

כדי לתמוך ב- labels האסמבלר מבצע שני מעברים על הקוד. במעבר הראשון זוכרים את הכתובות של כל ה- labels, ובמעבד השני בכל מקום שהיה שימוש ב- label בשדה ה- immediate, מחליפים אותו בכתובת ה- label בפועל כפי שחושב במעבר הראשון.

כמו כן שימו לב לשימוש ברגיסטר המיוחד $imm בהוראות השונות. למשל הוראת ה- beq בדוגמא קופצת במידה ואפס שווה לאפס. תנאי זה מתקיים תמיד ולכן זו בעצם שיטה לממש unconditional jump.

בנוסף להוראות הקוד, האסמבלר תומך בהוראה נוספת המאפשרת לקבוע תוכן של שורה ישירות בתמונת הזיכרון.

.word address data

כאשר address הינו כתובת המילה ו- data תוכנה. כל אחד משני השדות יכול להיות בדצימלי, או הקסאדצימלי בתוספת 0x. למשל:

```
.word 256 1                 # set MEM[256] = 1
.word 0x100 0x12345678      # MEM[0x100] = MEM[256] = 0x12345678
```

הנחות נוספות

ניתן להניח את ההנחות הבאות:

1. ניתן להניח שאורך השורה המקסימאלי בקבצי הקלט הוא 500.
2. ניתן להניח שאורך ה- label המקסימאלי הוא 50.
3. פורמט ה- label מתחיל באות, ואח"כ כל האותיות והמספרים מותרים.
4. צריך להתעלם מ- whitespaces כגון רווח או טאב. מותר שיהיו מספר רווחים או טאבים ועדיין הקלט נחשב תקין.
5. יש לתמוך בספרות הקסאדצימליות גם ב- lower case וגם upper case.
6. יש לעקוב אחרי שאלות, תשובות ועדכונים לפרויקט בפורום הקורס במודל.

דרישות הגשה

1. יש להגיש קובץ דוקומנטציה של הפרויקט, חיצוני לקוד, בפורמט pdf, בשם project1_id1_id2_id3.pdf כאשר id1,id2,id3 הם מספרי תעודת הזהות שלכם.

2. הפרויקט יכתב בשפת התכנות סי. האסמבלר והסימולטור הן תוכניות שונות, כל אחת תוגש בתיקייה נפרדת ותהיה פרויקט נפרד ב- visual studio, מתקמפלת ורצה בנפרד. <u>יש להקפיד שיהיו הערות בתוך הקוד המסבירות את פעולתו.</u>

3. יש להגיש את הקוד בסביבת - **visual studio community edition 2022** בסביבת windows (**שימו לב** – זה לא visual studio code) בכל תיקייה יש להגיש את קובץ ה- solution (עם סיומת sln.) ולוודא שהקוד מתקמפל ורץ, כך שניתן יהיה לבנות אותו ע"י לחיצה על build solution.

4. תוכניות בדיקה. הפרויקט שלכם יבדק בין השאר ע״י תוכניות בדיקה שלא תקבלו מראש, וגם ע״י ארבע תוכניות בדיקה שאתם תכתבו באסמבלי. יש להקפיד שיהיו הערות בתוך קוד האסמבלי. יש להגיש ארבע תוכניות בדיקה:

א. תוכנית sort.asm, אשר מבצעת מיון bubble sort של 16 מספרים בסדר עולה. המספרים נתונים בכתובות 0x100 עד 0x10f, ואלו גם כתובות המערך הממוין בסיום.

ב. תוכנית factorial.asm, המחשבת את !n (כלומר n עצרת) באופן רקורסיבי לפי האלגוריתם הבא. בתחילת הריצה n נתון בכתובת 0x100 והתוצאה תיכתב לכתובת 0x101. ניתן להניח כי n מספיק קטן כך שאין overflow.

```
int factorial(int n)
{
        if (n == 0)
                return 1;
        return n * factorial(n-1);
}
```

ג. תוכנית rectangle.asm, שמציירת על המסך מלבן מלא בצבע לבן (כל הפיקסלים בהיקף ובתוך שטח המלבן לבנים):

כאשר A הינו הקודקוד השמאלי עליון, B הינו הקודקוד השמאלי תחתון, C הקודקוד הימני תחתון, ו- D הקודקוד הימני עליון.

כתובות קודקודי המלבן יחסית לתחילת ה- frame buffer נתונות בכתובות 0x100 (A), 0x101 (B), 0x102 (C), 0x103 (D).

ד. תוכנית disktest.asm, שמבצעת סיכום של תוכן סקטורים 0 עד 3 בדיסק הקשיח וכותבת את תוצאת הסיכום לסקטור מספר 4, כלומר כל מילה בסקטור 4 תהיה סכום 4 המילים המתאימות מסקטורים 0 עד 3 :

```
for (i = 0;  i < 128; i++)
        sector4[i] = sector0[i] + sector1[i] + sector2[i]  + sector3[i]
```

את תוכניות הבדיקה יש להגיש בארבע תתי תיקיות שונות בשמות :

sort, factorial, rectangle, disktest

כל תיקייה תכיל את קבצי הקלט והפלט של ריצת תוכנית הבדיקה דרך האסמבלר והסימולטור. למשל בספרייה sort יהיו הקבצים הבאים :

sort.asm, memin.txt, diskin.txt, irq2in.txt, memout.txt, regout.txt, trace.txt, hwregtrace.txt, cycles.txt, leds.txt, display7seg.txt, diskout.txt, monitor.txt, monitor.yuv

חשוב לבדוק שגם האסמבלר וגם הסימולטור רצות מתוך חלון cmd ולא רק מתוך ה-visual. כמו כן חשוב לבדוק שאכן משתמשים בפרמטרים בשורת ההרצה ולא בשמות קבועים כיוון שאנחנו נבדוק את הקוד שלכם באמצעות בדיקות אוטומטיות שירוצו מקבצי batch מתוך חלון cmd עם קבצים שיכולים להיות בשמות אחרים.