

## אוניברסיטת תל-אביב, הפקולטה להנדסה

### פרויקט בקורס: מבנה המחשב 0512.4400

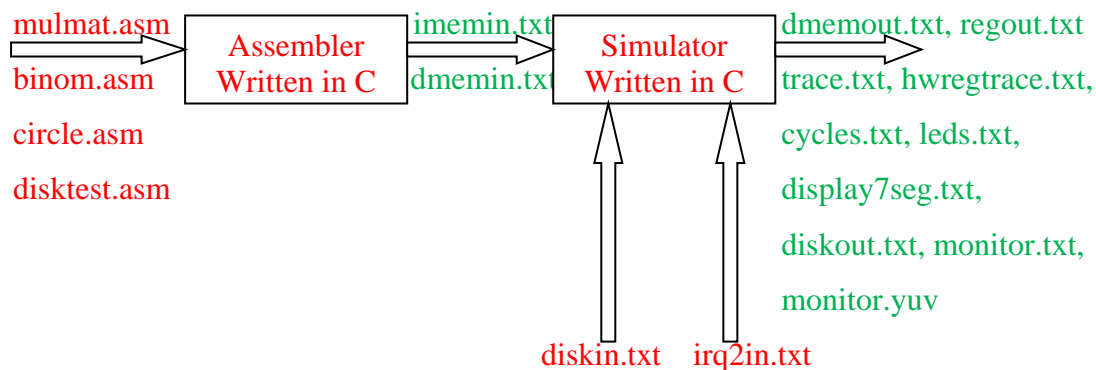
שנת הלימודים תשפ"ב, סמסטר א'

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. כל הוראה במעבד מתבצעת במחזור שעון אחד, והמעבד עובד בתדר של 512 הרץ - מבצע 512 הוראות אסמבלי בשנייה.

שימו לב: הסימולטור מסמלץ מעבד "איטי" שמבצע רק 512 הוראות אסמבלי בשנייה אבל אין צורך להאט את זמן הריצה האמיתי של הסימולטור על המחשב שלכם לקצב של מעבד ה-SIMP -- הסימולטור עצמו רץ על המעבד של המחשב האישי שלכם שעובד בתדר של גיגהרצים, ויכול לרוץ מאוד מהר ולסמלץ הרבה יותר מ-512 הוראות אסמבלי בשנייה של זמן "אמיתי" כפי שנמדד ע"י שעון יד למשל, חיצוני למחשב.

הדיאגרמה הבאה ממחישה את הפרויקט:



החלקים שאותם תכתבו בפרויקט ידנית מסומנים בצבע אדום, ואילו קבצי פלט שיוצרו 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	\$imm1	Sign extended immediate 1
2	\$imm2	Sign extended immediate 2
3	\$v0	Result value
4	\$a0	Argument register
5	\$a1	Argument register
6	\$a2	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 ובתרגולים עבור מעבד MIPS, בהבדל אחד: שני הרגיסטרים \$imm1, \$imm2, הינם רגיסטרים מיוחדים שלא ניתן לכתוב אליהם, ותמיד מכילים את שדות הקבועים - immediate1, immediate2, לאחר בצוע sign extension, כפי שקודד בהוראת האסמבלי. רגיסטר 0 הינו זהותית אפס. הוראות אשר כותבות ל- \$zero, \$imm1, \$imm2 לא משנות את ערכם.

### זיכרונות הוראות ונתונים

זיכרון ההוראות הינו ברוחב 48 סיביות ובעומק 4096 שורות. רגיסטר ה- PC לכן הינו ברוחב 12 ביטים, והוראות עוקבות מקדמות את PC באחד. זיכרון הנתונים הינו ברוחב 32 ביטים ובעומק 4096 שורות. הכתובת לזיכרון הנתונים היא לכן ברוחב 12 ביטים. בניגוד למעבד MIPS, למעבד SIMP אין תמיכה ב-byte או ב-short. כל גישה לזיכרון הנתונים קוראת או כותבת מילה ברוחב 32 ביטים.

## סט ההוראות וקידודם

למעבד SIMP יש פורמט בודד לקידוד כל ההוראות. כל הוראה הינה ברוחב 48 ביטים, כאשר מספרי הביטים של כל שדה נתונים בטבלה הבאה:

47:40	39:36	35:32	31:28	27:24	23:12	11:0
opcode	rd	rs	rt	rm	immediate1	immediate2

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

Opcode Number	Name	Meaning
0	add	$R[rd] = R[rs] + R[rt] + R[rm]$
1	sub	$R[rd] = R[rs] - R[rt] - R[rm]$
2	mac	$R[rd] = R[rs] * R[rt] + R[rm]$
3	and	$R[rd] = R[rs] \& R[rt] \& R[rm]$
4	or	$R[rd] = R[rs]   R[rt]   R[rm]$
5	xor	$R[rd] = R[rs] ^ R[rt] ^ R[rm]$
6	sll	$R[rd] = R[rs] \ll R[rt]$
7	sra	$R[rd] = R[rs] \gg R[rt]$ , arithmetic shift with sign extension
8	srl	$R[rd] = R[rs] \gg R[rt]$ , logical shift
9	beq	if ( $R[rs] == R[rt]$ ) pc = $R[rm]$ [low bits 11:0]
10	bne	if ( $R[rs] != R[rt]$ ) pc = $R[rm]$ [low bits 11:0]
11	blt	if ( $R[rs] < R[rt]$ ) pc = $R[rm]$ [low bits 11:0]
12	bgt	if ( $R[rs] > R[rt]$ ) pc = $R[rm]$ [low bits 11:0]
13	ble	if ( $R[rs] \leq R[rt]$ ) pc = $R[rm]$ [low bits 11:0]
14	bge	if ( $R[rs] \geq R[rt]$ ) pc = $R[rm]$ [low bits 11:0]
15	jal	$R[rd] = pc + 1$ (next instruction address), pc = $R[rm]$ [11:0]
16	lw	$R[rd] = \text{MEM}[R[rs]+R[rt]] + R[rm]$
17	sw	$\text{MEM}[R[rs]+R[rt]] = R[rm] + R[rd]$
18	reti	PC = IORegister[7]
19	in	$R[rd] = \text{IORegister}[R[rs] + R[rt]]$
20	out	IORegister $[R[rs]+R[rt]] = R[rm]$
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 512 byte 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 command
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.

כל מחזור שעון, המעבד בודק את הסיגנל:

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

במידה ו-  $irq == 1$ , 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  $PC = irqhandler$  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 timermax = timercurrent, 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 הפיקסלים 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 הפיקסלים 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 monitormcmd משמש עבור כתיבה של פיקסל. במחזור השעון שבו יש כתיבה monitormcmd=1 באמצעות הוראת out, מתבצע עדכון של הפיקסל שתוכנו ברגיסטר monitordata על המסך.  
קריאה מרגיסטר monitormcmd באמצעות הוראת in תחזיר את הערך 0.

## דיסק קשיח

למעבד SIMP מחובר דיסק קשיח בגודל 64 קילובייט, המורכב מ-128 סקטורים בגודל 512 בתים כל סקטור. הדיסק מחובר לפסיקה מספר 1, irq1, ומשתמש ב-DMA להעתקת הסקטור מהזיכרון לדיסק או להיפך.

תוכנו ההתחלתי של הדיסק הקשיח נתון בקובץ הקלט disk.in.txt, ותוכן הדיסק בסיום הריצה ייכתב לקובץ disk.out.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 איטרציה מביאים את ההוראה הבאה בכתובת ה-  $PC$ , מפענחים את ההוראה בהתאם לקידוד, ואח"כ מבצעים את ההוראה. בסיום ההוראה מעדכנים את  $PC$  לערך  $PC+1$  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 14 command line parameters לפי שורת ההרצה הבאה:

**sim.exe imemin.txt dmemin.txt diskin.txt irq2in.txt dmemout.txt regout.txt trace.txt hwregtrace.txt cycles.txt leds.txt display7seg.txt diskout.txt monitor.txt monitor.yuv**

הקובץ **imemin.txt** הינו קובץ קלט בפורמט טקסט אשר מכיל את תוכן זיכרון ההוראות בתחילת הריצה. כל שורה בקובץ מכילה תוכן שורה בזיכרון ההוראות, החל מכתובת אפס, בפורמט של 12 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 **dmemin.txt** הינו קובץ קלט בפורמט טקסט אשר מכיל את תוכן זיכרון הנתונים בתחילת הריצה. כל שורה בקובץ מכילה תוכן שורה בזיכרון הנתונים, 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, באותו פורמט כמו **dmemin.txt**, שמכיל את תוכן הדיסק הקשיח בתחילת הריצה.

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).

ארבעת 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 **dmemout.txt** הינו קובץ פלט, באותו פורמט כמו **dmemin.txt**, שמכיל את תוכן זיכרון הנתונים בסיום הריצה.

הקובץ **regout.txt** הינו קובץ פלט, שמכיל את תוכן הרגיסטרים  $R3-R15$  בסיום הריצה (שימו לב שאין להדפיס את הקבועים  $R0 - R2$ ). כל שורה תיכתב באותו פורמט כמו שורה ב- **dmemin.txt**, 8 ספרות הקסאדצימליות.

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



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

כל שדה מודפס בספרות הקסאדצימליות. ה- PC הינו ה- Program Counter של ההוראה (מודפס ב- 3 ספרות), ה- INST הינו קידוד ההוראה כפי שנקרא מהזיכרון (12 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. בשדות R1 ו-R2 יש לכתוב את תוכן הקבועים שנקראים מתוך ההוראה, לאחר ביצוע sign extension ל- 32 סיביות. למשל אם ביטים 11:0 בהוראה היו כולם אחדים, אז נכתוב עבור R2 את הערך FFFFFFFF.

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 בדצימאלי.

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.

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, בדצימאלי, 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, בדצימאלי, והמספר הימני הינו התצוגה, ב- 8 ספרות הקסאדצימליות.

הקובץ **diskout.txt** הינו קובץ פלט, באותו פורמט כמו **memin.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 הפיסקל בצד שמאל למעלה, והשורה האחרונה את הפיסקל בצד ימין למטה.

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-Y = color.

## 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 imemin.txt dmemin.txt
```

קובץ הקלט program.asm מכיל את תוכנית האסמבלי, קובץ הפלט imemin.txt מכיל את תמונת זיכרון ההוראות, וקובץ הפלט dmemin.txt מכיל את תמונת זיכרון הנתונים. קבצי הפלט של האסמבלר משמשים אח"כ כקבצי הקלט של הסימולטור.

כל שורת קוד בקובץ האסמבלי מכילה את כל 7 הפרמטרים בקידוד ההוראה, 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, rm, imm1, imm2
add $t3, $t2, $t1, $t0, 0, 0      # $t3 = $t2 + $t1 + $t0
add $t1, $t1, $imm1, $zero, 2, 0  # $t1 = $t1 + 2
add $t1, $imm1, $imm1, $imm1, 2, 0 # $t1 = 2 + 2 + 2 = 6
sub $t1, $imm1, $imm2, $imm2, 2, 3 # $t1 = 2 - 3 - 3 = -4
```

בכל הוראה, יש שלוש אפשרויות עבור שדות ה- immediate:

- ניתן לשים שם מספר דצימלי, חיובי או שלילי.
- ניתן לשים מספר הקסאדצימלי שמתחיל ב- 0x ואז ספרות הקסאדצימליות.
- ניתן לשים שם סימבולי (שמתחיל באות). במקרה זה הכוונה ל- label, כאשר label מוגדר בקוד ע"י אותו השם ותוספת נקודותיים.

דוגמאות:

```
bne $zero, $t0, $t1, $imm1, L1, 0      # if ($t0 != $t1) goto L1
                                         # (reg1 = address of L1)
add $t2, $t2, $imm1, $zero, 1, 0      # $t2 = $t2 + 1 (reg1 = 1)
beq $zero, $zero, $zero, $imm1, L2, 0  # jump to L2 (reg1 = address L2)

L1:
sub $t2, $t2, $imm1, $zero, 1, 0      # $t2 = $t2 - 1 (reg1 = 1)

L2:
add $t1, $zero, $imm1, $zero, L3, 0    # $t1 = address of L3
```

```

    beq $zero, $zero, $zero, $t1, 0, 0    # jump to the address specified in t
L3:
    jal $ra, $zero, $zero, $imm1, L4, 0    # function call L4, save return addr in
$ra
    halt $zero, $zero, $zero, $zero, 0, 0    # halt execution
L4:
    beq $zero, $zero, $zero, $ra, 0, 0    # return from function in address in $ra

```

כדי לתמוך ב- labels האסמבלר מבצע שני מעברים על הקוד. במעבר הראשון זוכרים את הכתובות של כל ה- labels, ובמעבר 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 בשורות. 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 מילה 32 ישירות בתמונת זיכרון הנתונים.

.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 0x1234ABCD # MEM[0x100] = MEM[256] = 0x1234ABCD

```

## 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 ב-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 בספרייה נפרדת, מתקמפלת ורצה בנפרד. יש להקפיד שיהיו הערות בתוך הקוד המסבירות את פעולתו.
3. יש להגיש את הקוד ב-visual studio בסביבת windows. בכל ספרייה יש להגיש את קובץ ה-solution, ולוודא שהקוד מתקמפל ורץ, כך שניתן יהיה לבנות אותו ע"י לחיצה על build solution. יש להגיש גם את ספריית ה-build כולל קובץ ה-executable הבנוי.
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 nulmat.asm, המבצעת כפל של שתי מטריצות בגודל 4x4 (כל איבר במטריצת התוצאה הינו מכפלה סקלרית של שורה ועמודה). ערכי המטריצה הראשונה נמצאים בכתובות 0x100 עד 0x10F, המטריצה השנייה בכתובות 0x110 עד 0x11F, ומטריצה התוצאה תיכתב לכתובות 0x120 עד 0x12F. ניתן להניח שאין overflow בחישוב.

כל מטריצה מסודרת בזיכרון לפי סדר שורות עולה, וכל שורה משמאל לימין. למשל עבור המטריצה הראשונה,  $a_{11}$  יהיה בכתובת 0x100,  $a_{12}$  בכתובת 0x101,  $a_{21}$  בכתובת 0x104 וכך הלאה.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

ב. תוכנית binom.asm, המחשבת את מקדם הבינום של ניוטון באופן רקורסיבי לפי האלגוריתם הבא. בתחילת הריצה n נתון בכתובת 0x100, k בכתובת 0x101, והתוצאה תיכתב לכתובת 0x102. ניתן להניח כי n מספיק קטן כך שאין overflow.

```
int binom(n, k)
{
    if (k == 0 || n == k)
        return 1;
    return binom(n-1, k-1) + binom(n-1, k);
}
```

ג. תוכנית circle.asm, שמציירת על המסך עגול מלא בצבע לבן (כל הפיקסלים בהיקף ובתוך שטח העגול), במרכז המסך. רדיוס העיגול (בפיקסלים) נתון בכתובת 0x100. ניתן להניח שהעגול נכנס במסך.

ד. תוכנית disktest.asm, שמבצעת הזזה של תוכן שמונת הסקטורים הראשונים בדיסק הקשיח (סקטורים 0 עד 7) סקטור אחד קדימה כך שבסיום הריצה, תוכן סקטור 1 יכיל את התוכן המקורי של סקטור 0 וכך הלאה עד תוכן סקטור 8 שיכיל את התוכן המקורי של סקטור 7.

5. את תוכניות הבדיקה יש להגיש בארבע ספריות בשמות:

mulmat, binom, circle, disktest

כל ספרייה תכיל עותק של קבצי ההרצה sim.exe, asm.exe וכמו כן את קבצי הקלט והפלט של ריצת תוכנית הבדיקה דרך האסמבלר והסימולטור. למשל בספרייה mulmat יהיו הקבצים הבאים:

sim.exe, asm.exe, mulmat.asm, imemin.txt, dmemin.txt, diskin.txt, irq2in.txt, dmemout.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.