

## אוניברסיטת תל-אביב, הפקולטה להנדסה

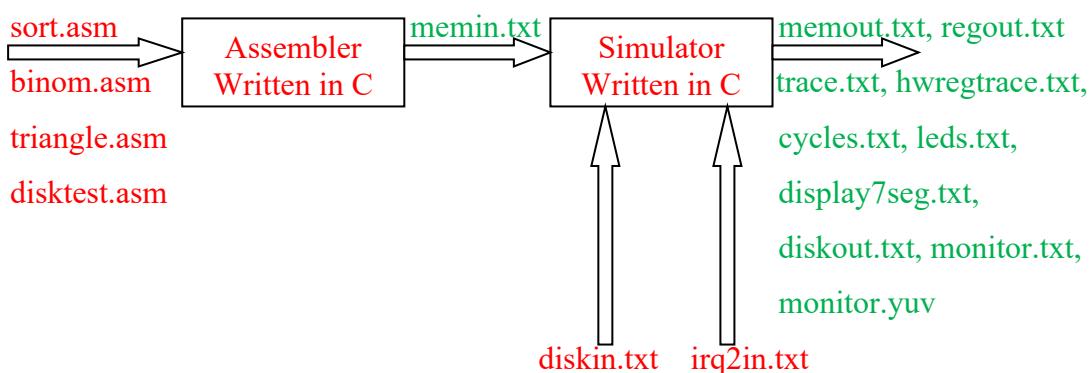
### פרויקט בקורס: **מבנה המחשב 0512.4400**

שנת הלימודים תשפ"ו, סטטוס א'

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

הסימולטור יסמל את מעבד ה-SIMP, וכן כן מספר התקני קלט/פלט: נורות, תצוגת-7, מוניטור מונוכרומטי ברזולוציה  $256 \times 256$ , ודייסק קשיח.

הдиגרמה הבאה ממחישה את הפרויקט:



החלקים הבאים כתבו בפרויקט ידנית מסומנים בכתב אדום, ואילו קבצי פלט שייצרו אוטומטית ע"י תוכנות האסmbller והסימולטור שתכתבו מסומנים בכתב ירוק.

## רגיסטרים

מעבד SIMP מכיל 16 רגיסטרים, שככל אחד מהם 32 ביבוב 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 ובתרגולים עבור מעבד MIPS, בהבדלים הבאים: הרגיסטרים \$zero, \$imm מיוחדים שלא ניתן לכתוב אליהם, ויכולים להיות בשימוש בכל מקום שבו יש שימוש ב- source operand :

הרגיסטר \$zero הינו זהותית אפס.

הרגיסטר \$imm מכיל את שדה הקבוע imm, כפי שказан בהוראת האסמבלי, לאחר בזע sign .extension

הוראות אשר כתובות ל- \$zero, \$imm לא משנהו את ערכם.

## **זיכרון ראשי, סט ההוראות וקידודם**

הזיכרון הראשי הינו ברוחב 20 סיביות ובעומק 4096 שורות. כל הוראה מתבצעת במחזור שעון בודד. למעבד SIMP יש שני פורמטים לקידוד ההוראות. הוראות אשר אין משתמשות בקבוע מקודדות בשורה אחת בזיכרון, כאשר מספרי הביטים של כל שדה נתונים בטבלה הבאה (פורמט :R

R format			
19:12	11:8	7:4	3:0
opcode	rd	rs	rt

במידה ויש שימוש בקבוע, ההוראה מקודדת בשתי שורות עוקבות (פורמט I).

I format			
19:12	11:8	7:4	3:0
opcode	rd	rs	rt
imm			

גיסטר ה- PC הינו ברוחב 12 ביטים. כדי להתקדם להוראה הבאה (במידה ואין קפיצה), PC מתתקדם באחד עבור ההוראות בפורמט R ללא הקבוע, או בשניים עבור ההוראות בפורמט I עם הקבוע.

האופקודות הנOMICים ע"י המעבד ומשמעותן 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	$R[rd] = R[rs] \ll R[rt]$
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[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	$R[rd] = \text{next instruction address}$ , pc = R[rs]
16	lw	$R[rd] = \text{MEM}[R[rs]+R[rt]]$ , with sign extension
17	sw	$\text{MEM}[R[rs]+R[rt]] = R[rd]$ (bits 19:0)
18	reti	PC = IORegister[7]
19	in	$R[rd] = \text{IORegister}[R[rs] + R[rt]]$
20	out	$\text{IORegister}[R[rs]+R[rt]] = R[rd]$
21	halt	Halt execution, exit simulator

## קלט/פלט

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

IORRegister 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>i</i> is on when <code>leds[i] == 1</code> , 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 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

## פסקות

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

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

פסיקה 2 מחוברת לקו חיצוני לمعالג irq. כובץ קלט לSIMULATOR קובע متى הפסיקה מתבצעת.

במחזור השעון בו הפסיקה מתתקבלת, מדליקים את אחד הרегистרים irq0status, irq1status, irq2status בהתאם. אם מספר פסיקות מתתקבלות באותו מחזור שעון, ידלקו בהתאם מספר רегистרי סטטוס.

בכל מחזור שעון המمعالג בודק את הסיגנל :

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

במידה ו-  $1 == \neg \text{irq}$ , והمعالג לא נמצא כרגע בתוך שגרת הטיפול בפסיקה, המمعالג קופץ לשגרת הטיפול בפסיקה שכותבתה בזיכרון נתונה ברегистר חומרה irqhandler. כלומר במחזור שעון זה מתבצעת הוראה ב- PC = irqhandler במקומו ב- PC המקורי. באותו מחזור שעון ה- PC המקורי נשמר לתוך רегистר חומרה irqreturn.

לעומת זאת במידה ו-  $1 == \text{irq}$  והمعالג עדיין נמצא בתוך שגרת הטיפול בפסיקה קוודמת (כלומר עדיין לא הריץ את הוראת ה- reti), המمعالג יתעלם, לא יקופץ וימשיך להריץ את הקוד כרגיל בתוך שגרת הפסיקה (כאשר המمعالג יחוור מהפסיקה, הוא יבדוק שוב את irq ואם יהיה צורך יקופץ שוב לשגרת הפסיקה).

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

.PC = irqreturn במאזעות הוראת reti, שתציב

## טี้ימר

מעבד SIMP תומר בטี้ימר של 32 ביטים, המחבר לפסיקה irq0. הוא מאפשר כאשר `.timerenable = 1`

ערך מונה הטี้ימר הנוכחי שמור ברגיסטר חומרה timercurrent. בכל מחזור שעון שבו הטี้ימר מאפשר, רגיסטר timercurrent מ庫ודם אחד.

במחזור השעון שבו irqstatus = timermax, מדליקים את irq0. במחזור שעון זה במקוםקדם 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 בית 0 מדליק/מכבה את נורה 0 (הימנית), ובית 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 לבן לבן לינארי.

במסך יש frame buffer פנימי בגודל 256x256 המכיל את ערכי הפייסקלים שכעת מוצגים על המסך. בתחלת העבודה כל הערכים מכילים אפס. הבפר מכיל שורות של 256 בתים שמתאימים לסריקת המסך מלמעלה למטה. כלומר שורה 0 בברפ' מכילה את הפייסקלים של השורה העליונה במסך. בכל שורה סריקת הפייסקלים במסך הינה משמאליימין.

רגיסטר monitoraddr מכיל את האופטט בברפ' 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 בAlamat הוראת out, מתבצע עדכון של הפייסקל שתוכנו ברגיסטר monitordata על המסך.

קריאה מריגיסטר monitorcmd בAlamat הוראת to תחזיר את הערך 0.

## דיסק קשיח

למעבד SIMP מחובר דיסק קשיח המורכב מ- 128 סקטורים, כאשר כל סקטור מכיל 128 שורות ברוחב 20 ביטים. הדיסק מחובר לפסיקה מס' 1, irq1, ומשתמש ב-DMA להעתיקת הסקטור מהזיכרונו לדיסק או להיפך.

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

לפני מתן הוראות 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 שני רגיסטרים אלו מוחתכים, נויתנים הוראות כתיבה או קריאה ע"י כתיבה לרגיסטר חומרה diskcmd.

זמן הטיפול של הדיסק בהוראות קריאה או כתיבה הוא 1024 מחזורי שעון. במהלך זמן זה יש להעתיק את תוכן הבפר לדיסק במידה והיתה 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 ישנו רגיסטר ה-diskcmd לערך 0, והדיסק יודיע על פסיקה ע"י הדלקת irqstatus1.

## הסימולטור

הסימולטור מסמלץ את לולאת ה- PC.fetch-decode-execute . בתחילת הריצה PC=0. בכל איטרציה מביאים את הוראה הבאה בכתובת ה- PC, מפענחים את הוראה בהתאם לקידוד, ואח"כ מבצעים את הוראה. בסיום הוראה מעדכנים את PC לערך PC+1 או PC+2 (כתלות בהאם יש שימוש בקבוע בהוראה), 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 לתוכן 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 בתחלת הריצה. כל שורה בקובץ מכילה תוכן שורה בזיכרון, החל מכתובת אפס, בפורמט של 5 ספרות הקסדצימליות. 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 בתחלת הריצה, כאשר כל שורה מכילה 5 ספרות הקסדצימליות. 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 שביהם קו הפסיק החיצוניirq2 עולה ל- 1, כל מחזור שעון כזה בשורה נפרדת בסדר עולה.uko כל פעם עולה ל- 1 למחזור שעון בודד ואז יורד חזרה לאפס (אלא אם כן מופיעה שורה נוספת בקובץ עבור מחזור השעון הבא). שלושת קבועי הקלט צריכים להיות קיימים לפחות פעם אחת בכלם אין בהם שימוש (לדוגמא גם עבור קוד אסטראלי שאינו משתמש בדיסק הקשיח יהיה קיים קבוע 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 R1-R15-R2-R15 בסיום הריצה (שים לב שאין להדפיס את הקבועים R1 – R0). כל שורה תיכתב ב- 8 ספרות הקסדצימליות.

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 הקסדצימליות. ה- PC הינו ה- Program Counter של הוראה (מודפס ב- 3 ספרות), ה- INST הינו קידוד התא הראשון של הוראה כפי שנקרה מהזיכרון (5 ספרות),

ואח"כ יש את תוכן הרегистרים **לפניהם** ביצוע ההוראה (כלומר את תוצאת הביצוע ניתן לראות רק ברגיסטרים של השורה הבאה). כל רגיסטר מודפס ב- 8 ספרות.  
בשדה R0 יש כתוב 8 אפסים. בשדה R1 יש כתוב 0 עבור הוראה מסוג R, או את תוכן הקבוע לאחר ביצוע sign extension ל- 32 סיביות עבור הוראה מסוג I.  
הקובץ **hwregtrace.txt** הינו קובץ פלט, המכיל שורת טקסט עבור כל קריאה או כתיבה לרегистר חומרה (באמצעות הוראות **mo** ו- **ou**) בפורמט הבא :

#### CYCLE READ/WRITE NAME DATA

כאשר השדה CYCLE הוא מספר מחזורי השעון בדצימלי.  
השדה הבא מכיל READ או WRITE בהתאם להאם קוראים או כותבים לרегистר החומרה.  
השדה NAME מכיל את שם רегистר 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 שרצתה התוכנית.  
הקובץ **leds.txt** מכיל את סטטוס 32 הנורות. בכל מחזורי שעון שאחת הנורות משתנה (נדלקת או נכנית), כותבים שורה עם שני מספרים ורוחם ביןיהם : המספר השמאלי הינו מחזורי השעון, בדצימלי, והמספר הימני מצב כל 32 הנורות ב- 8 ספרות הקסאדצימליות.  
הקובץ **display7seg.txt** מכיל את תצוגת ה- 7-segment display. בכל מחזורי שעון שהתצוגה משתנה, כותבים שורה עם שני מספרים ורוחם ביןיהם : המספר השמאלי הינו מחזורי השעון, בדצימלי, והמספר הימני התצוגה, ב- 8 ספרות הקסאדצימליות.  
הקובץ **diskout.txt** הינו קובץ פלט, אליו פורמט כמו **diskin.txt**, שמכיל את תוכן הדיסק הקשיח בסיום הרצפה.  
הקובץ **monitor.txt** מכיל את ערכי הפלטטים שבמסך בסיום הרצפה. כל שורה מכילה ערך פיסקל בודד (8 ביטים) בשתי ספרות הקסאדצימליות, כאשר סריקת המסך היא מלמעלה למטה, ומשמאלי לימין. לדוגמה השורה הראשונה בקובץ מכילה את ערך הפיקסל מצד שמאל למעלה. במידה ומספר השורות בקובץ קטן ממספר הפלטטים במסך, ההנחה היא ששאר הפלטטים מכילים אפס.  
הקובץ **yuv.yuv** הינו קובץ בינארי אשר מכיל את אותו דата כמו **monitor.txt**, ונitin להציגו על המסך באמצעות התוכנה **yuvplayer**:

<https://github.com/Tee0125/yuvplayer>

כasher בפרמטרים בוחרים י- size = 256x256 .color = Y

## האסמלר

כדי שייהי נוח לתוכנת את המעבד וליצור את תמונה הזיכרון בקובץ memin.txt, נכתב בפרויקט גם את תוכנית האסמלר. האסמלר יכתב בשפת סי, ויתרגם את תוכנית האסמלר שכתובה בטקסט בשפת אסמלרי, לשפת המכוונה. ניתן להניח שקובץ הקלט תקין.

בדומה לSIMULATOR, האסמלר הינו command line application עם שורת הריצה הבאה:  
asm.exe program.asm memin.txt

קובץ הקלט program.asm מכיל את תוכנית האסמלרי, קובץ הפלט memin.txt מכיל את תמונה הזיכרון הראשי, ומשמש Ach"c כקובץ קלט לSIMULATOR.

כל שורת קוד בקובץ האסמלרי מכילה את כל 5 הפרמטרים בקידוד ההוראה, כאשר הפרמטר הראשון הינו האופקודה, והפרמטרים מופרדים ע"י סימני פסיק. לאחר הפרמטר האחרון מותר להוסיף את הסימן # והערה מצד ימין, לדוגמה:

```
# 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 הקסדצימלי שמתחליל ב- x0 ואז ספרות הקסדצימליות.
- ניתן לשים שם סימבולי (שמתחליל באוט). במקרה זה הכוונה ל-label, כאשר label מוגדר בקוד ע"י אותו השם ותווסף נקודותתיים.

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 האסמבילר מבצע שני מעברים על הקוד. במעבר הראשון זורקרים את הכתובות של כל ה- labels, ובמעבר השני בכל מקום שהיה שימוש ב- label בשדה ה- immediate, מחליפים אותו בכתובת ה- label בפועל כפי שהוא במעבר הראשון. כמו כן שימוש לב לשימוש ברגיסטר המယוד \$imm בהוראות השונות. למשל הוראת ה- beq בדוגמה קופצת במידה ואפס שווה לאפס. תנאי זה מתקיים תמיד ולכן זו בעצם שיטה למש .unconditional jump.

בנוסף להוראות הקוד, האסמבילר תומך בהוראה נוספת המאפשרת לקבוע תוכן של שורה ישרות בתמונה הזיכרון.

.word address data

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

.word 256 1 # set MEM[256] = 1

.word 0x100 0x1234A # MEM[0x100] = MEM[256] = 0x1234A

## הנחות נוספות

ניתן להניח את ההנחות הבאות:

1. ניתן להניח שאורך השורה המקסימלי בקבצי הקלט הוא 500.
2. ניתן להניח שאורך ה- **label** המקסימלי הוא 50.
3. פורמט ה- **label** מתייחס באותיות, ואח"כ כל האותיות והמספרים מותרים.
4. נדרש להעתלם מ- **whitespaces** כגון רווח או טאב. מותר שהיו מספר רווחים או טאים ועדין הקלט נחשב תקין.
5. יש לתמוך בספרות הקסדצימליות גם ב- **lower case** וגם **upper case**.
6. יש לעקוב אחרי שאלות, תשובה ועדכונים לפרויקט בפורום הקורס במודול.

## דרישות הגשה

1. יש להגיש קוד דוקומנטציה של הפרויקט, חיצוני לקוד, בפורמט **pdf**, בשם **project1\_id1\_id2\_id3.pdf** כאשר **id1,id2,id3** הם מספרי תעודת זהות שלכם.
2. הפרויקט יכתב בשפת התכנות סי. האסמבילר והסימולטור הן תוכניות שונות, כל אחת תוגש בתיקייה נפרדת ותהיה פרויקט נפרד ב- **visual studio**, מתكمפלת ורצת בנפרד.  
יש להקפיד שה יהיו הערכות בתוך הקוד המסבירות את פעולתו.
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 קוד ה- **sln** (עם סיוםת **.sln**) ולודא שהקוד מתكمפל ורץ, כך שניתן יהיה לבנות אותו ע"י **.build solution** לחיצה על **build**.

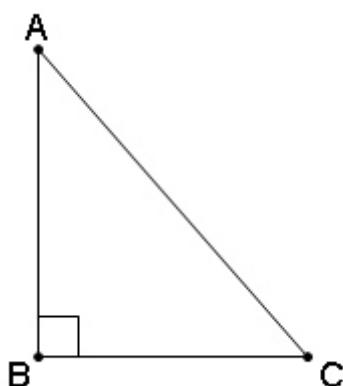
4. תוכניות בדיקה. הפרויקט שלכם יבדק בין השאר ע"י תוכניות בדיקה שלא תקבלו מראש, וגם ע"י ארבע תוכניות בדיקה שאתם כתבו באסמבלי. יש להקפיד שהיו העורות בתוך קוד האסמבלי. יש להגשים ארבע תוכניות בדיקה:

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

ב. תוכנית binom.asm, המחשבת את מקדם הבינום של ניוטון באופן רקורסיבי לפי האלגוריתם הבא. בתחילת הרצה מ נתון כתובות 0x100,k בכתובת 0x100, וההתוצאה תיכתב לכתובת 0x101. ניתן להניח כי ממספריק 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 triangle.asm, שמצוירת על המסך משולש ישר זווית מלא בצלע לבן (כל הפיקסלים בהיקף ובתוך שטח המשולש לבנים):



כאשר A הינו קודקוד השמאלי עליון, זווית B היא ישרה, ו- C הינו קודקוד הימני-תחתון. צלע AB הינה אנכית, וצלע BC אופקית. כתובות הקודודי המשולש יחסית לתחילת ה-frame buffer נתונות כתובות .0x100 (A), 0x101 (B), 0x102 (C)

ד. תוכנית disktest.asm, שמבצעת סיכום של תוכן סקטוריים 0 עד 7 בדיסק הקשיך וכותבת את תוצאת הסיכום לסקטור מס' 8, כלומר כל מילה בסקטור 8 תהיה סכום 8 המילים המתאימות מסקטוריים 0 עד 7 :

```
for (i = 0; i < 128; i++)  
    sector8[i] = sector0[i] + sector1[i] + ... + sector7[i]
```

את תוכניות הבדיקה יש להגיש באربע תת-תיקיות שונות בשםות :

sort, binom, triangle, disktest

כל תיקייה תכיל את קבצי הקלט והפלט של ריצת תוכנית הבדיקה דרך האסמלר 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 האסמלר וגם הסימולטור רצות מתוך חלון cmd ולא רק מתוך ה-visual. כמו כן חשוב לבדוק שגם משתמשים בפרמטרים בשורת ההרצה ולא בשמות קבועים כיון שאנו נבדוק את הקוד שלכם באמצעות בדיקות אוטומטיות שירוץ מקבצי batch מתוך חלון cmd עם קבצים שימושיים להיות בשםות אחרים.