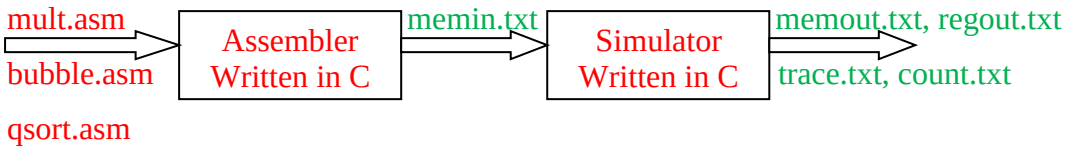
אוניברסיטת תל-אביב, הפקולטה להנדסה

פרויקט ISA בקורס: מבנה המחשב 0512.4400

שנת הלימודים תשע"ח, סמסטר א'

בפרויקט נתרגל את נושא שפת המחשב, 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. 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. מספרים שליליים מיוצגים במשלים ל- 2. שמות הרגיסטרים, מספרם, ותפקיד כל אחד מהם בהתאם ל- calling conventions, נתונים בטבלה הבאה:

Register Number	Register Name	Purpose
0	$zero	Constant zero
1	$at	Assembler temporary
2	$v0	Result value
3	$a0	Argument register
4	$a1	Argument register
5	$t0	Temporary register
6	$t1	Temporary register
7	$t2	Temporary register
8	$t3	Temporary register
9	$s0	Saved register
10	$s1	Saved register
11	$s2	Saved register
12	$gp	Global pointer (static data)

13	$sp	Stack pointer
14	$fp	Frame pointer
15	$ra	Return address

שמות הרגיסטרים ותפקידם דומים למה שראינו בהרצאה ובתירגולים עבור מעבד MIPS.
רגיסטר 0 הינו זהותית אפס. הוראות אשר כותבות ל- $zero לא משנות את ערכו.

רוחב מילה וזיכרון ראשי

בניגוד למעבד MIPS, למעבד SIMP אין תמיכה ב- byte או ב- short. המילה ברוחב 32 סיביות, וכך גם הזיכרון הראשי. כל הקריאות והכתיבות מהזיכרון הראשי הן תמיד של 32 ביטים בבת אחת. לכן כתובות הזיכרון הראשי יהיו ביחידות של מילים ולא בתים כמו ב- MIPS.
כלומר כתובות עוקבות בזיכרון יתקדמו ב- 1 ולא ב- 4.
אין גם שאלה של big endian או little endian כי תמיד עובדים ביחידות של מילה שלמה. כמו כן רגיסטר ה- Program Counter (PC) מתקדם באופן רגיל (אם לא קופצים) רק ב- 1, ולא ב- 4.
מרחב הכתובות של הזיכרון הראשי במעבד SIMP הוא ברוחב 16 סיביות בלבד (65536 מילים).

סט ההוראות וקידודם

למעבד SIMP יש פורמט בודד לקידוד כל ההוראות. כל הוראה הינה ברוחב 32 ביטים, כאשר מספרי הביטים של כל שדה נתונים בטבלה הבאה:

31:28	27:24	23:20	19:16	15:0
opcode	rd	rs	rt	imm (קבוע)

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

Number	Name	Meaning
0	add	R[rd] = R[rs] + R[rt]
1	sub	R[rd] = R[rs] – R[rt]
2	and	R[rd] = R[rs] & R[rt]
3	or	R[rd] = R[rs] | R[rt]
4	sll	R[rd] = R[rs] << R[rt]
5	sra	R[rd] = R[rs] >> R[rt], arithmetic shift with sign extension
6	limm	R[rd] = sign extended imm (bits 31:16=sign, bits 15:0=imm)
7	beq	if (R[rs] == R[rt]) pc = imm
8	bgt	if (R[rs] > R[rt]) pc = imm
9	ble	if (R[rs] <= R[rt]) pc = imm

10	bne	if (R[rs] != R[rt]) pc = imm
11	jal	R[15] = pc + 1 (next instruction address), pc = imm
12	lw	R[rd] = MEM[R[rs]+imm]
13	sw	MEM[R[rs]+imm] = R[rd]
14	jr	pc = R[rd]
15	halt	Halt execution, exit simulator

הסימולטור

הסימולטור הינו פונקציונאלי, כלומר ללא צורך לסמלץ זמנים אלא רק את פעולת התוכנית. 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 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 חמישה command line parameters לפי שורת ההרצה הבאה:

sim memin.txt memout.txt regout.txt trace.txt count.txt

הקובץ **memin.txt** הינו קובץ קלט בפורמט טקסט אשר מכיל את תוכן הזיכרון הראשי בתחילת הריצה. כל שורה בקובץ מכילה תוכן מילה בזיכרון, החל מכתובת אפס, בפורמט של 8 ספרות הקסאדצימליות. במידה ומספר השורות בקובץ קטן מ- 65536, ההנחה הינה ששאר הזיכרון מעל הכתובת האחרונה שאותחלה בקובץ, מאופס. ניתן להניח שקובץ הקלט תקין.

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 R0-R15 בסיום הריצה. כל שורה תיכתב באותו פורמט כמו שורה ב- memin.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 הינו 8 ספרות הקסאדצימליות. ה- PC הינו ה- Program Counter של ההוראה, ה- INST הינו קידוד ההוראה כפי שנקרא מהזיכרון, ואח"כ יש את תוכן הרגיסטרים **<u>לפני</u>** ביצוע ההוראה (כלומר את תוצאת הביצוע ניתן לראות רק ברגיסטרים של השורה הבאה). בשדה R0 יש לכתוב 8 אפסים.

הקובץ **count.txt** הינו קובץ פלט, שמכיל את מספר ההוראות שבוצעו ע"י התוכנית.

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 program.asm mem.txt

קובץ הקלט program.asm מכיל את תוכנית האסמבלי, וקובץ הפלט mem.txt מכיל את תמונת הזיכרון. קובץ הפלט של האסמבלר משמש אח"כ כקובץ הקלט של הסימולטור.

כל שורת קוד בקובץ האסמבלי מכילה את כל 5 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, imm
limm $t0, $zero, $zero, 2      # $t0 = 2
limm $t1, $zero, $zero, -1     # $t1 = -1 = 0xFFFFFFFF
add $t2, $t1, $t0, 0           # $t2 = $t1 + $t0 = -1 + 2 = 1
```

בכל הוראה, יש שלוש אפשרויות עבור שדה ה- imm:

- 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, $t1, L1          # if ($t0 != $t1) goto L1
        limm $t1, $zero, $zero, 0x1      # $t1 = 1
        add $t2, $t2, $t1, 0             # $t2 = $t2 + $t1
        beq $zero, $zero, $zero, L2      # unconditional jump to L2
L1:
        sub $t2, $t2, $t1, 0             # $t2 = $t2 – $t1
L2:
        limm $t1, $zero, $zero, L3       # $t1 = address of L3
        jr $t1, $zero, $zero, 0          # jump to the address specified in t1
L3:
```

```
        jal $zero, $zero, $zero, L4      # function call L4, save return addr in $ra
        halt $zero, $zero, $zero, 0      # exit simulator

L4:
        jr $ra, $zero, $zero, 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, 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 להוראת ה- beq בדוגמא אשר 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 בזיכרון. הוראה זו מאפשרת לקבוע דאטא בקובץ תמונת 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 257 -1                 # set MEM[257]  = -1 (0xFFFFFFFF)
.word 0x100 0x1234ABCD       # MEM[0x100] = MEM[256] = 0x1234ABCD
```

האסמבלר ממלא את תוכן תמונת הזיכרון בערך ההוראה word. ברגע שהיא נקראת. אם יש מספר אתחולים לאותה הכתובת (או ע"י הוראות word. או ע"י הוראות אסמבלי לאותה כתובת), האתחול האחרון קובע.

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. ב- linux אפשר להשתמש בהוראה man כדי לקבל הסברים (manual page) על פונקציות ספרייה של שפת סי. למשל man strtok או man sscanf. אפשר גם לחפש בגוגל למשל "strtok manual page".

6. יש לעקוב אחרי שאלות, תשובות ועדכונים לפרוויקט בפורום הקורס במודל.

דרישות הגשה

1. יש להגיש קובץ דוקומנטצייה של הפרויקט, חיצוני לקוד, בפורמט pdf.
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.

אם משתמשים ב- visual studio בסביבת windows, יש להגיש את קובץ ה- solution עבור כל אחת מהספריות, ולוודא שהקוד מתקמפל ורץ, כך שניתן יהיה לבנות אותו ע"י לחיצה על build solution.

אם משתמשים בסביבת linux, יש להוסיף קובץ Makefile עבור כל ספרייה כך שניתן יהיה לבנות את הקוד ע"י הרצת make.

3. תוכניות בדיקה. הפרויקט שלכם יבדק בין השאר ע"י תוכניות בדיקה שלא תקבלו מראש, וגם ע"י שלוש תוכניות בדיקה שאתם תכתבו באסמבלי. יש להגיש כל תוכנית בדיקה בספרייה נפרדת, המכילה את קובץ האסמבלי program.asm, את קובץ ה- memin.txt שנוצר ע"י האסמבלר שאותו הרצתם על הקוד, ואת קבצי memout.txt, regout.txt, trace.txt, count.txt שנוצרו ע"י הסימולטור.

יש לכתוב את קוד האסמבלי תוך הקפדה על הקונבציות המקובלות שראיתם בהרצאות ובתירגולים (מחסנית גודלת כלפי כתובות נמוכות, לשמור רגיסטרים שמורים למחסנית, להעביר פרמטרים לפונקצייה ב- $a, להחזיר ערך ב- $v, וכו'). את ערך רגיסטר ה- $sp המצביע לראש המחסנית יש לאתחל בתחילת הריצה לערך 2048 (0x800).

יש להקפיד שיהיו הערות בתוך קוד האסמבלי.

יש להגיש שלוש תוכניות בדיקה:

א. תוכנית multtable.asm, הכותבת את לוח הכפל לזיכרון החל מכתובת 1024 כך ש- MEM[1024+(j-1)*10+(i-1)] = j*i עבור i,j בין 1 לבין 10 כולל.

ב. תוכנית bubble.asm, הממשת מיון של מערך מספרים בסדר יורד ע"י שימוש באלגוריתם bubble sort. מערך המספרים שאותו יש למיין נמצא בתאים 1024-1039.

ג. תוכנית mergesort.asm, הממשת מיון של מערך מספרים בסדר יורד ע"י שימוש באלגוריתם merge sort. מערך המספרים שאותו יש למיין נמצא בתאים 1024-1039.