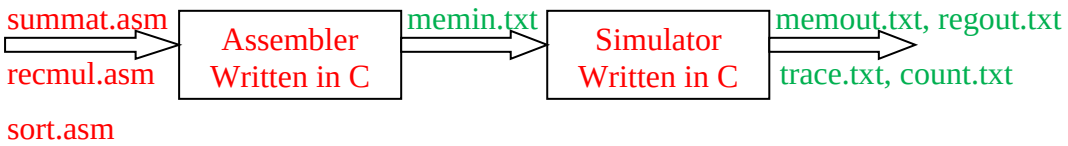
אוניברסיטת תל-אביב, הפקולטה להנדסה

פרויקט ISA בקורס: מבנה המחשב 0512.4400

שנת הלימודים תשע"ט, סמסטר ב'

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 ברוחב 16 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 במילים ברוחב 32 סיביות. המילה ברוחב 16 סיביות, וכך גם הזיכרון הראשי. כל הקריאות והכתיבות מהזיכרון הראשי הן תמיד של 16 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 12 סיביות בלבד (4096 מילים).

סט ההוראות וקידודם

למעבד SIMP יש 3 פורמטים לקידוד ההוראות, אשר נבדלים ברוחב שדה הקבוע. כל הוראה הינה ברוחב 16 ביטים, כאשר מספרי הביטים של כל שדה נתונים בטבלה הבאה:

15:12	11:8	7:4	3:0
opcode	rd	rs	imm4
opcode	rd	imm8	
opcode	imm12		

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

<table>
<tr><th>Number</th><th>Name</th><th>Meaning</th></tr>
<tr><td>0</td><td>noimm</td><td>
<table>
<tr><th colspan="10">subopcode (value of imm4)</th></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>other</td></tr>
<tr><td>Add</td><td>sub</td><td>and</td><td>or</td><td>sll</td><td>srl</td><td>sra</td><td>mul</td><td>jr</td><td>rsv</td></tr>
</table>
if (imm4 == add) R[rd] = R[rd] + R[rs]
if (imm4 == sub) R[rd] = R[rd] - R[rs]
if (imm4 == and) R[rd] = R[rd] & R[rs]
if (imm4 == or) R[rd] = R[rd] | R[rs]
if (imm4 == sll) R[rd] = R[rd] << R[rs]
if (imm4 == srl) R[rd] = R[rd] >> R[rs] (logical shift with zero extension)
if (imm4 == sra) R[rd] = R[rd] >>> R[rs] (arithmetic shift with sign extension)
if (imm4 == mul) R[rd] = R[rd] * R[rs] (low 16 bits of the result)
if (imm4 == jr) pc = R[rd] & 0xfff
other values of imm4 reserved
</td></tr>
<tr><td>1</td><td>beq</td><td>if (R[rd] == R[rs]) pc = pc + 1 + simm4</td></tr>
<tr><td>2</td><td>bne</td><td>if (R[rd] != R[rs]) pc = pc + 1 + simm4</td></tr>
<tr><td>3</td><td>lw</td><td>R[rd] = MEM[(R[rs]+simm4) & 0xfff]</td></tr>
<tr><td>4</td><td>sw</td><td>MEM[(R[rs]+simm4) & 0xfff] = R[rd]</td></tr>
<tr><td>5</td><td>bgtz</td><td>if (R[rd] > 0) pc = pc + 1 + simm8</td></tr>
<tr><td>6</td><td>blez</td><td>if (R[rd] <= 0) pc = pc + 1 + simm8</td></tr>
<tr><td>7</td><td>limm</td><td>R[rd] = simm8</td></tr>
<tr><td>8</td><td>lhi</td><td>R[rd] = (imm8 << 8) | (R[rd] & 0xff)</td></tr>
<tr><td>9</td><td>j</td><td>pc = imm12</td></tr>
<tr><td>10</td><td>jal</td><td>R[15] = (pc + 1) & 0xfff (next instruction address),
pc = imm12</td></tr>
<tr><td>11-14</td><td>reserved</td><td>reserved instructions</td></tr>
<tr><td>15</td><td>halt</td><td>Halt execution, exit simulator</td></tr>
</table>

simm4, simm8, simm12 הינן הקבועים imm4, imm8, imm12 לאחר בצוע sign extension ל- 16 ביטים.

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 בשם **sim.exe** אשר מקבל חמישה command line parameters לפי שורת ההרצה הבאה:

sim.exe 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 4 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 **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, 4 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 4 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 4 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 בשם **asm.exe** עם שורת ההרצה הבאה:

```
asm.exe 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 3 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, imm
limm $t0, $zero, 2          # $t0 = 2
limm $t1, $zero, -1         # $t1 = -1 = 0xFFFF
noimm $t1, $t0, add         # $t1 = $t1 + $t0 = -1 + 2 = 1
```

בכל הוראה, יש ארבע 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 את ה- subopcode עבור הוראות עם opcode = noimm במקום ערך מספרי, כדי שהקוד יהיה יותר קריא. ניתן לכתוב את ה- subopcode באותיות קטנות או גדולות.
- ניתן לשים שם סימבולי (שמתחיל באות), ושונה מאחד ה- subopcodes. במקרה זה הכוונה ל- label, כאשר label מוגדר בקוד ע"י אותו השם ותוספת נקודותיים.

דוגמאות:

```
    bne $t0, $t1, L1            # if ($t0 != $t1) goto L1
    limm $t1, $zero, 0x1        # $t1 = 1
    noimm $t2, $t1, add         # $t2 = $t2 + $t1
    j $zero, $zero, L2          # unconditional jump to L2
L1:
```

```
        noimm $t2, $t1, sub          # $t2 = $t2 – $t1
L2:
        limm $t1, $zero, L3          # $t1 = address of L3
        noimm $t1, $zero, jr         # jump to the address specified in t1
L3:
        jal $zero, $zero, L4          # function call L4, save return addr in $ra
        halt $zero, $zero, 0         # exit simulator

L4:
        noimm $ra, $zero, jr         # 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.

בנוסף להוראות הקוד, האסמבלר תומך בהוראה נוספת המאפשרת לקבוע תוכן של מילה 16 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 128 1              # set MEM[128] = 1
.word 129 -1             # set MEM[129]  = -1 (0xFFFF)
.word 0x80 0xABCD        # MEM[0x80] = MEM[128] = 0xABCD
```

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. 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.pdf כאשר id1 ו- id2 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. יש לוודא שהפרויקט עובד על הגרסא האחרונה של visual studio שניתנת להורדה מהאתר של מיקרוסופט ולהתקנה על המחשב האישי, או לחילופין על הגרסא המותקנת במעבדת המחשבים בפקולטה.

5. תוכניות בדיקה. הפרויקט שלכם יבדק בין השאר ע״י תוכניות בדיקה שלא תקבלו מראש, וגם ע״י שלוש תוכניות בדיקה שאתם תכתבו באסמבלי.
יש לכתוב את קוד האסמבלי תוך הקפדה על הקונבנציות 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 1024 (0x400).
יש להקפיד שיהיו הערות בתוך קוד האסמבלי.

יש להגיש שלוש תוכניות בדיקה:

א. תוכנית addmat.asm, המבצעת סכום של שתי מטריצות בגודל 4x4. 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 recmul.asm, המחשבת כפל שני מספרים 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 recmul(n, k)
{
        if (n == 0)
                return 0;
        return recmul(n-1, k) + k;
}
```

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

6. את תוכניות הבדיקה יש להגיש בספרייה בשם tests, המכילה 5 קבצים:

sim.exe, asm.exe, summat.asm, recmul.asm, sort.asm

בנוסף יהיו בתוך הספרייה tests שלוש תתי-ספריות בשמות summat, recmul, sort שבהם יהיו תוצאות ההרצה של כל אחד מהטסטים:

- קובץ ה- memin.txt שנוצר ע"י האסמבלר שאותו הרצתם על הקוד.
- הקבצים memout.txt, regout.txt, trace.txt, count.txt שנוצרו ע"י הסימולטור.

למשל כדי ליצור את הקבצים בתוך summat תריצו בחלון cmd (לחצו על start, run ואז רשמו cmd) מתוך ספריית ההגשה הראשית שלכם את רצף הפקודות:

```
cd tests
mkdir summat
cd summat
..\asm.exe ..\summat.asm memin.txt
..\sim.exe memin.txt memout.txt regout.txt trace.txt count.txt
```

חשוב לבדוק שהשורות הכתובות מעלה רצות מתוך חלון cmd ולא רק מתוך ה- visual כיוון שאנחנו נבדוק את הקוד שלכם באמצעות בדיקות אוטומטיות שירוצו מקבצי batch מתוך חלון cmd.