מבנה מחשבים

מיני-פרויקט 2

<u>תאריך הגשה</u>: 12/05/22 בשעה 23:55

שאלה מעשית: תמיכה בפקודות נוספות בSingle Cycle MIPS.

לתרגיל הבית צורף קובץ בשם single_cycle_mips.circ. קובץ זה מכיל מימוש logisim של המעבד אשר נלמד בכיתה במהלך תרגול 7.

בכל הסעיפים הבאים עליכם להוסיף למעבד תמיכה בפקודות נוספות על ידי שינויים ביחידות הcontrol, מסלולי הנתונים או הALU. אין צורך לבצע שינויים בקובץ הרגיסטרים, בPC או ברכיבי הזכרון.

עבור כל סעיף אתם נדרשים לממש את הפתרון על גבי המעבד הנתון ולהגיש אותו כקובץ בשם single_cycle_mips_<index>.circ כאשר index הוא מספר הסעיף. לדוגמא, עבור סעיף 3, הוסיפו להגשה קובץ בשם single_cycle_mips_3.circ.

בנוסף, עבור כל סעיף, הוסיפו תיאור קצר של השינויים שביצעתם וכל מידע נוסף הנדרש לגבי קידוד הפקודה בסעיפים בהם ניתן חופש קידוד (למשל אם נדרש funct מסוים). את התיאור יש להוסיף לקובץ הPDF המכיל את התשובות התיאורטיות.

מותר להשתמש בכלל הרכיבים הניתנים בLogisim, עם כל מספר של כניסות או רוחב אותות.

רכיבים שיכולים להיות שימושיים במיוחד הם הmux, shifter, bit extender והsplitter.

עבור סעיפים 1-6, השתמשו בקוד ומבנה הinstruction האמיתיים של כל פקודה.

תוכלו למצוא אותם ב Instruction Set Reference המצורף לתרגיל או בדף הסיכום של אוניברסיטת קורנל:

https://www.cs.cornell.edu/courses/cs3410/2017sp/resources/3410_MIPS.pdf

עבור סעיף 7 אתם מוזמנים להשתמש בכל opcode או קידוד פקודה פנוי.

ממשו תמיכה בפקודות הבאות:

1) addi
2) j (jump)
3) jr
4) bne
5) sll

<u>שימו לב לקידוד המיוחד של פקודת sll. בפרט, שימו לב להבדל ברגיסטר ממנו נלקח המספר המוסת.</u>

6) ori

<u>שימו לב כי בפקודות מסוג זה נעשה שימוש בzero extension ולא בsign extension.</u>

7) jnezr המוגדרת באופן הבא:

```
jnezr $s0, $s1
```

ופירושה: if ($s1 != 0) ? PC =$s0 : PC = PC+4

את נכונות הפתרון שלכם תוכלו לבדוק על ידי קידוד ישיר של קוד המכונה המתאים לinstruction set.

לחלופין, (עבור סעיפים 1-6) תוכלו לממש קטע קוד assembly המשתמש בפקודות ולהמירו לקוד מכונה ע"י שימוש בMARS או SPIM:

http://courses.missouristate.edu/KenVollmar/mars/index.htm

http://spimsimulator.sourceforge.net

שני הסימולטורים יציגו לכם את הקידוד ההקסדצימלי של כל שורת קוד, אותו תוכלו לטעון ידנית לinstruction memory.

על מנת לערוך את זכרון הפקודות יעמדו לרשותכם שתי אפשרויות:

1) עריכה ישירה על ידי פתיחת הinstruction memory, לחיצה ימנית על רכיב הזכרון ובחירת 'edit contents'.
2) טעינה של תמונת זכרון על ידי פתיחת הinstruction memory, לחיצה ימנית על רכיב הזכרון ובחירת 'load image'.

באופן דומה ניתן לערוך גם את קובץ הזכרון.

לנוחיותכם, לתרגיל צורפו זוג קבצים (example_instructions.mem ו example_data.mem) המכילים תמונת זכרון של סט הפקודות הבא:

```
lw $s1, 0($zero)
lw $s2, 4($zero)
add $s0, $zero, $zero
loop:
add $s0, $s0, $s1
slt $t0,  $s2, $s0
beq $t0, $zero, loop
sw  $s0, 8($zero)
```

וכן ערכי זכרון התחלתיים.

שימו לב:

1) SPIM מניח מימוש של פקודות branch ביחס לPC ולא ל PC+4, לכן צריך לחסר 1 מקידוד הOFFSET שהוא נותן. MARS משתמש בקידוד הנלמד בכיתה.
2) עקב הגבלות של Logisim, הן זכרון הפקודות והן זכרון המידע ממומשים בעזרת כתובות של 24 ביט. רכיבי הזכרון כפי שהם נתונים לכם יקחו רק את 24 הLSB של כל כתובת. בנוסף, כאשר אתם ממלאים את הפקודות בזכרון, יש למלא אותן בכל כניסה רביעית. ראו לדוגמא את הקבצים המצורפים.

☺ בהצלחה! ☺