פרוייקטי סוף בארכיטקטורה ומבנה מחשבים

1. מחשב ה Multi Cycle MIPS

א. ממשו ב VHDL את מחשב ה Multi Cycle MIPS. ניתן להשתמש בארכיטקטורה מבנית והתנהגותית במימוש שלכם.

לצורך הבהירות נגדיר את רגיסטרי העזר ב MultiCycle MIPS:

A, B, ALUOut, Memory Data Register, Instruction Register

לרשימה זו ניתן להוסיף גם את רגיסטר ה PC.

הבהרות למימוש שלכם:

- יש לממש את המחשב לפי נספח תרשימי החומרה הצמוד לקובץ זה באתר הקורס.
- חובה לממש קודם מודולים חשובים כמו Controller, ALU, Memory, Register File, כל אחד לחוד. הבקר הוא מכונת מצבים מסוג Moore.
- לכל קו בקרה שהבקר מוציא חובה שיהיה סיגנל משלו.
- לא חובה לממש בוררים כמודולים נפרד.
- לא חובה לממש רגיסטרי עזר כמודולים נפרדים. עם זאת במימוש שלכם חייבים להופיע סיגנלים עבור כל רגיסטר עזר שיש בחומרה, כולל ה PC.
- לא חובה לממש יחידת הרחבת סימן כמודול נפרד.
- לא חובה לממש יחידת הזזה שמאלה ב2 ביט כמודול נפרד.
- שאר חלקי החומרה, כגון שערי OR או AND ימומשו בצורה הנוחה לכם.

ב. כתבו תוכנית שמחשבת 100 איברי סדרת פיבונצ'י ראשונים וכותבת אותם במערך המוקצה בזיכרון. שני האיברים הראשונים הם a0=1, a1=1 והם לא מופיעים במערך בתחילת ריצת התוכנית. הפכו את התוכנית לקודים הקסדצימליים ושמרו אותה בקובץ שייטען לזיכרון של ה Multi Cycle MIPS.

ג. כתבו testbench שמריץ את התוכנית של סעיף ב. הראו צורות גלים שמראות שהתוכנית מתבצעת כנדרש והציגו את תוכן המערך בזיכרון לפני ואחרי ריצת התוכנית.

2. מחשב ה Pipelined MIPS

א. ממשו ב VHDL את מחשב ה MIPS המצונר ללא טיפול ב Hazards בחומרה.

לצורך הבהירות נגדיר את רגיסטרי העזר ב Pipelined MIPS:

IF/ID, ID/EX, EX/MEM, MEM/WB

לרשימה זו ניתן להוסיף גם את רגיסטר ה PC.

שימו לב להנחיות/הבהרות הבאות:

- יש לממש את המחשב לפי נספח תרשימי החומרה הצמוד לקובץ זה באתר הקורס, עמוד 6 ללא יחידות הקידומים וה Hazard Detection Unit.
- שימו לב ששלב ה Branch Resolution הוא שלב ID ולא שלב מאוחר יותר. יש לממש את החומרה בהתאם.
- חובה לממש קודם מודולים עיקריים כמו Controller, ALU, Data/Instruction Memory, Register File, PC+4 Adder, Branch Address Adder, כל אחד לחוד.
- לכל קו בקרה שהבקר מוציא חובה שיהיה סיגנל משלו. הסיגנל ייקרא בשם אחר בכל שלב של הצינור, למשל RegWrite_ID, Regwrite_EX, Regwrite_MEM, RegWrite_WB עבור הקו RegWrite בכל השלבים שבהם הוא קיים. כמובן שקו הבקרה ישפיע רק בשלב ה WB.
- הבקר חייב להוציא קן בקרה Branch עבור פקודת beq.
- יש להוסיף תמיכה גם בפקודת j שתסתיים בשלב ה ID בדומה לפקודת beq.
- לא חובה לממש בוררים כמודולים נפרד.

- לא חובה לממש רגיסטרי עזר כמודולים נפרדים. עם זאת במימוש שלכם חייבים להופיע סיגנלים עבור כל רגיסטר עזר שיש בחומרה, כולל ה PC.
- לא חובה לממש יחידת הרחבת סימן כמודול נפרד.
- לא חובה לממש יחידת הזזה שמאלה ב2 ביט כמודול נפרד.
- שאר חלקי החומרה, כגון שערי OR או AND ימומשו בצורה הנוחה לכם.
- אין להתייחס במימוש שלכם לסיכוני נתונים, סיכוני Load או סיכוני Branch, אבל בתוכנית שתרוץ על המחשב תצטרכו להכניס nops, להרחיק את הפקודות התלויות זו מזו או להשתמש ב Branch Delay Slots, אם מתאפשר, לפי הצורך.

ב. כתבו תוכנית שמקבלת מערך של 20 מספרים בזיכרון, למשל המערך הבא:

4, 5, 12, 19, 2, 18, 3 ,4, 9, 10, 17, 15, 11, 7 ,6, 8, 1, 20, 16, 14

יש למיין את המערך על ידי שימוש ב Insersion sort בסדר עולה.

בתוכנית יש לטפסל ב Hazards על ידי שינוי סדר פקודות, שימוש ב Branch Delay Slots או הוספת nops.

המטרה שלכם להשתמש במספר המינימלי של nops שמתאפשר.

ג. כתבו testbench שמריץ את התוכנית של סעיף ב. הראו צורות גלים שמראות שהתוכנית מתבצעת כנדרש והציגו את תוכן המערך בזיכרון לפני ואחרי ריצת התוכנית.

3. תכנון מסלול נתונים ובקר (למציאת המספר הראשוני הקרוב ביותר הקטן ממספר נתון)

יש לתכנן מסלול נתונים ובקר עבור מערכת שבהינתן קלט חיובי שלם n ($2 \le n$) מוצאת את המספר הראשוני הגדול ביותר שקטן מ n, למשל עבור n=20, יוחזר 19 ועבור n=17 יוחזר 13.

מסלול הנתונים מורכב מרגיסטרים (ניתן לקחת יותר מ 2), יחידת ALU עם שני אופרנדים A ו B ומוצא Out, עורק נתונים משותף (bus שמחובר לכניסת הנתונים החיצונית של המערכת n, לכניסות הרגיסטרים ולמוצא ה ALU), חוצצי tri-state ובוררים לפי הצורך. מוצא המערכת ייצא ממוצא אחד הרגיסטרים.

- ה ALU יודע לחשב את הפעולות הבאות בהתאם לכניסה Op שלה:

Op	פעולה שמתבצעת ב ALU
00	Out = A-1
01	Out = A mod B שארית החלוקה של A ב B
10	Out = A
11	לצירוף זה ניתן לבחור איזו פעולה תתבצע לפי הצורך, אבל רק פעולה אריתמטית פשוטה כמו חיבור או לוגית פשוטה כמו OR, AND, XOR. ניתן גם לא להשתמש בערך זה של Op.

- לא חובה להשתמש בכל הצירופים של Op עבור ה ALU.
- ה ALU גם מוציא קו Zero שדלוק כאשר תוצאת החישוב שלו היא אפס.

- למסלול הנתונים כולו יש כניסת נתונים אחת מהמשתמש החיצוני בשם n לקבלת הקלט ויציאת נתונים אחת בשם res להוצאת הפלט אל המשתמש החיצוני. רוחב הכניסה והיציאה צריך להיות פרמטר של המערכת עם ערך ברירת מחדל של 16 ביט.
- הבקר הוא מכונת מצבים מסוג Moore/Mealy (לבחירתכם) עם מינימום מצבים אפשרי. לבקר יש כניסת שעון clk, כניסת reset לאיפוס וכניסת start להתחלת החישוב. יש לו יציאה בשם done שעולה כשהחישוב מסתיים והתוצאה מוכנה.
- שאר יציאות הבקר אל מסלול הנתונים (קווי הבקרה) לבחירתכם.

- למערכת כולה יש לכן כניסת נתונים אחת n, יציאת נתונים אחת res, כניסת שעון clk, כניסת reset, כניסת start, יציאת done.

א. תכננו ושרטטו את מסלול הנתונים הנדרש לביצוע האלגוריתם.
תכננו ושרטטו את דואגרמת המצבים של הבקר השולט במסלול הנתונים. יש לתכנן אותו למינימום מצבים.

ב. יש לכתוב מודול עבור ה ALU, עבור כל מסלול הנתונים ועבור הבקר. כעת יש לחבר את מסלול הנתונים והבקר לקבלת המודול הסופי.
לא חובה לכתוב מודול נפרד עבור רגיסטרים, בוררים, חוצצי tri state, אבל חובה לייצג בקוד מוצא של כל אחד מהם על ידי סיגנל ייעודי.

ג. יש לממש testbench למערכת כולה ולהראות שהמערכת עובדת על 7 קלטים שונים הכוללים את הערכים הבאים: n=17, 20, 37, 47, 57. הראו את התוצאות של המערכת ואת מספר מחזורי השעון לכל חישוב (בטבלה).

ד. ממשו מסלול נתונים ובקר נוספים עבור אותה הבעיה, כאשר מסלול הנתונים כולל שתי יחידות ALU ורגיסטרים, בוררים וחוצצי tri state לפי הצורך. חשבו כיצד ניתן לזרז את החישובים על ידי שימוש בשתי יחידות חישוביות וממשו את המערכת בהתאם. מטרתכם להגיע לביצוע המהיר ביותר.

ה. כתבו testbench למערכת החדשה שמכניס לה את אותם הקלטים כמו בסעיף ג. הראו גם כאן את התוצאות של המערכת ואת מספר מחזורי השעון לכל חישוב (בטבלה).
פי כמה בממוצע המימוש השני מהיר יותר מהראשון? בהנחה שעלות החומרה במימוש זה כפולה מזה של המימוש הראשון, האם המימוש השני משתלם?

4. זיכרון מטמון

א. יש לממש ב VHDL מודול של זיכרון ראשי שניתן לכתוב אליו ולקרוא ממנו ערכים הקסדצימליים. גודל הזיכרון הוא 1KByte, כאשר רוחב כל כתובת הוא בית אחד.
יש לממש גם מודול של זיכרון נוסף, זיכרון המטמון, המסודר בצורת Direct mapped. גודל ה cache הוא הוא של 8 בלוקים, כאשר גודל כל בלוק הוא 4byte.

- ה cache עובד בצורת LRU, Write Back, לכן יש לשמור לכל בלוק גם סיביות Modified, Valid, LRU לניהול תקין של זיכרון המטמון.

ב. חברו את מודול הזיכרון עם מודול ה cache ליצירת מודול היררכיית זיכרון שבכל גישה לכתובת מסויימת בזיכרון ניגש קודם כל לבדוק אם הבלוק שלה קיים ב cache (במקרה של קריאה או כתיבה). אם לא קיים הבלוק בזיכרון המטמון, ניגש לזיכרון להביא אותו אליו ורק אז ניגש אליו.
יש לשים לב להערות הבאות:

- הגישה לזיכרון היא ברמת כתובת, אבל בפועל בודקים תמיד אם כל הבלוק של אותה כתובת נמצא ב cache ומביאים/כותבים לזיכרון הראשי תמיד בלוק שלם.
- מדיניות Write Back במקרה כתיבה פירושה שבמקרה כתיבה מעדכנים את הבלוק בזיכרון המטמון ללא עדכון של הזיכרון הראשי עד שמפנים את הבלוק מזיכרון המטמון לזיכרון הראשי. סיבית ה Modified מסמנת לכל בלוק אם כשנפנה אותו נצטרך לכתוב אותו לזיכרון הראשי.
- מדיניות LRU פירושה שכשנדרש פינוי של בלוק מה cache, נפנה את הבלוק שניגשנו אליו הכי מזמן. לניהול תקין של מדיניות זו יש לשמור זמן גישה אחרונה לבלוק עבור כל בלוק, אבל קיימים פתרונות פשוטים יותר (Pseudo LRU) שניתן להשתמש בהם.

ג. כתבו test bench עבור מודול היררכיית הזיכרון שמדמה לפחות 100 גישות לאותם בלוקים ברצף (כ 10 בלוקים או יותר, בדומה לתוכנית הרצה בלולאה וניגשת לאותו מערך שוב ושוב) ובנוסף גישות לבלוקים שאינם ברצף (מספרי בלוקים שרירותיים לבחירתכם). מדדו את ה hit rate של הגישות.

ד. שנו את זיכרון המטמון בלבד להיות באותו גודל של 8 בלוקים (כל בלוק 4bytes), אבל מסודר בצורת Fully Associative. יצרו מודול היררכיית זיכרון שמשתמש ב cache זה במקום הקודם.

כתבו testbench שמבצע את אותן הגישות של סעיף ג לזיכרון ומודד hit rate.
כיצד השתנה ה hit rate? איזה משני המימושים של ה cache יעיל יותר?