Project Tomasulo

|  |  |
| --- | --- |
| ארכיטקטורת מחשבים | |
| סמסטר א' תשע"ד | |
| לירון גורן | 038214409 |
| טל דויד | 200729267 |

Table of Contents

[הנחות מקדימות 3](#_Toc377246941)

[מבנה הנתונים 4](#_Toc377246942)

[Package: main 4](#_Toc377246943)

[Parser 4](#_Toc377246944)

[Memory 4](#_Toc377246945)

[Instruction 4](#_Toc377246946)

[Global 4](#_Toc377246947)

[Tomasulo 4](#_Toc377246948)

[Package: buffers 5](#_Toc377246949)

[Buffers 5](#_Toc377246950)

[LoadStoreBuffer 5](#_Toc377246951)

[LoadBuffer & StoreBuffer 5](#_Toc377246952)

[Package: reservationStations 7](#_Toc377246953)

[ReservationStations 7](#_Toc377246954)

[ReservationStation 7](#_Toc377246955)

[AluReservationStation & MulOrAddReservationStation 7](#_Toc377246956)

[Package: units 8](#_Toc377246957)

[Unit 8](#_Toc377246958)

[integerALU & LoadStore & FPAddSub & FPMul 8](#_Toc377246959)

[Package: registers 8](#_Toc377246960)

[Registers 8](#_Toc377246961)

[Register 8](#_Toc377246962)

[IntRegister & FloatRegister 8](#_Toc377246963)

[Package: exceptions 8](#_Toc377246964)

[ה- Flow של התוכנית 9](#_Toc377246965)

[השדות שבמחלקה Tomasulo 9](#_Toc377246966)

[אתחול 9](#_Toc377246967)

[מהלך התוכנית 11](#_Toc377246968)

[Step() 11](#_Toc377246969)

[Issue() 11](#_Toc377246970)

[branchResolution() 11](#_Toc377246971)

[Execute() 13](#_Toc377246972)

[writeToCDB() 13](#_Toc377246973)

[פונקציות תומכות של המחלקה Tomasulo 14](#_Toc377246974)

# הנחות מקדימות

* הגדרנו את ה- issue\_cycle של הוראת HALT כמחזור שעון שבו פוענחה הוראת ה- HALT (סיימנו את המימוש לפני עדכון ה- trace של הדוגמה שפורסמה באתר). עם זאת, כמובן שהתוכנית לא מסתיימת עד שלא כל ההוראות שקדמו ל- HALT סיימו את שלב ה Write To CDB.
* אנו לא ממתינים מחזור שעון לריקון תור ההוראות במקרה של הוראת קפיצה (סיימנו את המימוש לפני עדכון ה- trace של הדוגמה).
* בהוראות store, למרות שלא כותבים את הערך המוחזר ל- CDB, עדיין "מבזבזים" מחזור שעון אחד לפני פינוי הבאפר. כלומר אם הוראת store מבצעת execute במחזורי שעון 5-10, במחזור 11 נרוקן את הבאפר שלה, ולא במחזור 10.

כלומר, בהתאם להנחות הללו, קובץ ה trace שלנו תואם את הראשון שפורסם באתר בטרם העדכונים.

* סדר הכניסה של ההוראות ליחידות הפונקציונאליות נקבע על פי הסדר של ההוראות שנכנסו ל- executeList: ההוראה הראשונה שהייתה מוכנה לבצע execute תיכנס ראשונה, וכך הלאה.
* פענוח קפיצות מבוצע בשלב ה- issue. כל עוד הרגיסטרים לא מוכנים, לא נבצע issue להוראות הבאות בתור.
* אנו מניחים שהקלט לתוכנית הוא סינטקטית תקין וכן שהפורמט של הקבצים תקין.

# מבנה הנתונים

## Package: main

### Parser

מטרת ה- Parser היא לפרסר את קבצי ה- input בתחילת התוכנית וליצור את קבצי ה- output בסופה, זאת באמצעות מתודות אשר נקראות מה – main לאחר יצירת אובייקט ה Parser.

### Memory

Class זה מייצג את זיכרון התוכנית. מכיל מערך בגודל 1024 של integers המייצג את הזיכרון. בנוסף מכיל מתודות של load / store עבור פקודות הדורשות גישה לזיכרון, ומתודות נוספות המשמשות ליצירתו לאחר תהליך הפרסור, בעת יצירת אובייקט ה Tomasulo.

### Instruction

מייצג פקודה בתוכנית. מכיל את השדות הבאים:

* שדות הפקודה – OPCODE, DST, SRC0, SRC1, IMM.
* מספר המחזור שבו התבצע שלב מסוים של הפקודה: issue\_cycle, execute\_start\_cycle, execute\_end\_cycle, write\_to\_CDB\_cycle.
* תוצאת ביצוע הפקודה – result. למשל עבור add, כאשר הפקודה תסיים את שלב ה- execute, שדה ה- result יתעדכן עם התוצאה של ה- add. יכול להיות int או float.
* binaryInst – הפקודה בייצוג בינארי.
* Station – ה- reservation station / buffer שבו הכנסנו את הפקודה (לא רלוונטי לפקודות מסוג branch או jump).
* PC – ה- PC של הפקודה.

ל- class זה מספר פונקציות תומכות:

* freeStation – כאשר הפקודה הסתיימה וביצענו כתיבה ל- CDB, מתודה זו תשחרר את ה- reservation station / buffer הרלוונטי לפקודה ותהפוך אותו ל- Not busy.
* isReadyToBeExecuted – מחזיר TRUE כאשר הפקודה מוכנה לעבור משלב ה- issue לשלב ה- execute, כלומר כאשר האופרנדים של ה- reservation station / buffer הרלוונטי מוכנים.

### Global

מחזיק מספר קבועים סטטיים של התוכנית.

### Tomasulo

זהו ה- flow המרכזי של התוכנית, יפורט בהמשך.

## Package: buffers

### Buffers

Class זה מייצג את כל ה- load & store buffers שיש למעבד. באתחול מקבל את מספר הבאפרים מכל סוג על פי קובץ הקונפיגורציה של המעבד ומאתחל לפי זה את הבאפרים. פונקציות תומכות:

* getLoadBuffer(String name) – מחזיר load buffer שאינו busy. במתודה זו נשתמש בשלב ה- issue.
* getFreeStoreBuffer(String name) – מחזיר store buffer שאינו busy. במתודה זו נשתמש בשלב ה- issue.
* isThereFreeLoadBuffer – מחזיר true אם יש load buffer פנוי. במתודה זו נשתמש בשלב ה- issue.
* isThereFreeStoreBuffer – מחזיר true אם יש store buffer פנוי. במתודה זו נשתמש בשלב ה- issue.
* updateTags – עובר על כל הבאפרים, ואם יש באפר אשר תלוי ב- tag הרלוונטי, מעדכן את ה- value. במתודה זו נשתמש בשלב ה- write to CDB.
* isThereStoreAddressCollision – עוברים על כל ה load & store buffers, ואם קיים באפר עסוק עם הוראה בעלת pc נמוך יותר, אשר פונה לאותה כתובת בזיכרון, או לחלופין עדין לא חושבה עבורה הכתובת, נחזיר TRUE. למעשה לפי המימוש שבחרנו ייתכן שנחזיר כי קיימת התנגשות עם כתובת באפר אחר, למרות שיתברר לנו בהמשך לאחר חישוב הכתובת של הבאפר האחר, כי לא קיימת התנגשות ורק אז נחזיר FALSE, לאחר שכביכול ויתרנו על מספר מחזורי שעון "לשווא".
* isThereLoadAddressCollision – בדומה למתודה האחרונה, רק שעבור פקודות load אנו מחפשים התנגשות כתובות רק עם ה store buffers.
* freeBuffer(String station) – מתודה זו מקבלת את שם הבאפר המפונה, שולפת אותו ממבנה הנתונים ומפעילה דרכו את המתודה free אשר מחזירה את שדה ה busy ל IDLE, וכן מבצעת השמה של מחרוזות ריקות לשדות ה tag של הבאפר, למטרת השימוש העתידי. במתודה זו נשתמש בשלב ה- write to CDB.

### LoadStoreBuffer

Class זה מייצג באפר באופן כללי. יורש את ReservationStation ובכך מקבל את תכונותיו. מכיל את הכתובת המחושבת שאליה נעשה store או שממנה נעשה load, וכן שני משתנים המייצגים את value1 – הערך שיש ב- intRegisters[SRC0] ו- value2 – עבור store בלבד – הערך שנרצה לשמור בזיכרון הנמצא ב- floatRegisters[SRC1]. פונקציות תומכות:

* calculateAddress – מחשב את הכתובת הרלוונטיים על פי הערך שניתן מהרגיסטר וה- imm של הפקודה.
* setValue1 – מעדכן את value1 וכן את ה- tag שלו להיות empty.
* setValue2 – מעדכן את value2 וכן את ה- tag שלו להיות empty.

### LoadBuffer & StoreBuffer

Classes אלו מייצגים load & store buffers. הם יורשים מ- LoadStoreBuffer.

## Package: reservationStations

### ReservationStations

Class זה מכיל את כל ה-reservation stations שיש למעבד. באתחול מקבל את מספר ה- reservation stations מכל סוג על פי הקונפיגורציה של המעבד, ומאתחל בהתאם את ה- reservation stations. פונקציות תומכות:

* isThereFreeAluRS – מחזיר true אם יש ALU reservation station שאינו busy. במתודה זו נשתמש בשלב ה- issue.
* isThereFreeMulRS – מחזיר true אם יש MUL reservation station שאינו busy. במתודה זו נשתמש בשלב ה- issue.
* isThereFreeAddRS – מחזיר true אם יש ADD reservation station שאינו busy. במתודה זו נשתמש בשלב ה- issue.
* getFreeAluReservationStation – מחזיר ALU reservation station שאינו busy. במתודה זו נשתמש בשלב ה- issue.
* getFreeMulReservationStation – מחזיר MUL reservation station שאינו busy. במתודה זו נשתמש בשלב ה- issue.
* getFreeAddReservationStation – מחזיר ADD reservation station שאינו busy. במתודה זו נשתמש בשלב ה- issue.
* updateTags(String station, Object object) – עובר על כל ה- reservation stations, ואם יש reservation station אשר תלוי ב- tag הרלוונטי, מעדכן את ה- value. במתודה זו נשתמש בשלב ה- write to CDB.
* freeReservationStation(String station) – מתודה זו מקבלת את שם התחנה המפונה, שולפת אותה ממבנה הנתונים ומפעילה דרכה את המתודה free אשר מחזירה את שדה ה busy ל IDLE, וכן מבצעת השמה של מחרוזות ריקות לשדות ה tag של התחנה, למטרת השימוש העתידי. במתודה זו נשתמש בשלב ה- write to CDB.

### ReservationStation

Class זה מייצג reservation station. מכיל את ה- opcode, ואינדיקציה בנוגע להאם התחנה במצב busy.

מימשנו את מחלקה זו כמממשת את מנשק Comparable מתוך מטרה לדרוס את מתודת compareTo. המחלקה הקודמת שדיברנו עליה, ReservationStations, שומרת במשתנים פרטיים רשימות של תחנות העבודה משלושת הסוגים השונים. עבור המתודות שהוזכרו: isThereFreeAddRS, isThereFreeMulRS, isThereFreeAluRS, אנחנו קוראים למתודה הסטטית Collections.sort עם הרשימה הרלוונטית, כאשר מה שקורה למעשה הוא שהמתודה הסטטית הזאת ממיינת את הרשימה באמצעות אותה מתודת compareTo שדרסנו במחלקה ReservationStation. המיון הראשוני מבוצע לפי שדה ה BUSY, כאשר כל התחנות הפנויות יופיעו כראשונות, והמיון המשני הוא מיון לפי שם התחנה.

פונקציות תומכות:

* isReady – מחזיר true אם ה- instruction הרלוונטי לאותה תחנה מוכן לביצוע execute.
* Free – משחרר את ה- reservation station.

### AluReservationStation & MulOrAddReservationStation

אלו הם בעצם ה- reservation stations עצמם, בסוגם השונים. Classes אלו יורשים מ- ReservationStation.

## Package: units

### Unit

Class זה מייצג יחידה במעבד. מכיל את ה- delay של היחידה ואת מספר ה- instructions המחכים ליחידה בתור, על מנת לדעת מתי instruction מסויים יכול להיכנס ליחידה.

### integerALU & LoadStore & FPAddSub & FPMul

אלו הן היחידות עצמם, כולן יורשות מ- Unit. מכילות פונקציית execute: כאשר instruction מסויים נכנס ליחידה על מנת לבצע execute, נשתמש בפונקציה זו.

## Package: registers

### Registers

Class זה מכיל את כל הרגיסטרים שבתוכנית. מכיל פונקציות לעדכון הרגיסטרים / שליפה מהרגיסטרים, ולעדכון ה- tags כאשר כותבים ל- CDB.

### Register

מייצג רגיסטר בודד. מכיל את הסטאטוס של הרגיסטר: האם הוא תלוי ב- tag מסויים או שיש בו ערך מעודכן.

### IntRegister & FloatRegister

Classes אלו יורשים מ- Register, ומייצגים רגיסטרים מסוג int או float.

## Package: exceptions

ב- package זה מימשנו exceptions שונים הנקראים מתוך התוכנית במקרה שצריך לזרוק חריג:

* AddressForLoadStoreOutOfBoundException
* InvalidDelayValueForUnitsException
* MissingNumberOfLoadStoreBuffersException
* MissingNumberOfReservationStationsException
* ProgramCounterOutOfBoundException
* UnknownOpcodeException

# ה- Flow של התוכנית

כל ה- flow של התוכנית מתבצע במחלקה Tomasulo, ומשתמש במבני הנתונים שבנינו.

## השדות שבמחלקה Tomasulo

* instructionsQueue – מכיל את כל הפקודות הנכנסות לתור בשלב ה- issue.
* instructionsStaticQueue – מכיל את כל הפקודות הנכנסות לתור בשלב ה- issue, אבל הוא לא מתרוקן, שכן הוא נועד להדפסת ה- trace בסיום התוכנית.
* waitingList – מכיל את כל הפקודות שביצעו issue, אבל עדיין אינן מוכנות לעבור לשלב ה- execute.
* executeList – מכיל את כל הפקודות שכעת בשלב ה- execute או מוכנות לבצע execute (מחכות ליחידה הרלוונטית שתתפנה).
* writeToCDBList – מכיל את כל הפקודות שסיימו execute ומוכנות לכתוב ל- CDB (פקודות store לא ייכנסו לרשימה זו).
* Memory – הזיכרון.
* Registers – הרגיסטרים של המעבד.
* reservationStations – כל ה- reservation stations של המעבד.
* Buffers – כל הבאפרים של המעבד.
* fetchingStatus – האם עלינו לבצע fetch.
* globalStatus – האם התוכנית הסתיימה.
* fetchedHaltInst – האם ביצענו fetch לפקודת HALT.
* Clock
* Pc
* Alu\_unit – יחידת ה- ALU של המעבד.
* FP\_add\_sub\_unit – יחידת החיבור / חיסור floats של המעבד.
* FP\_mult\_unit – יחידת הכפל floats של המעבד.
* Load\_store\_unit – יחידת ה- load / store של המעבד.

## אתחול

באתחול ה- Tomasulo אנחנו נאתחל את כל מה שהוזכר לעיל בהתאם לקונפיגורציה ולזיכרון.

## מהלך התוכנית

כל עוד התוכנית לא הסתיימה, נקרא לפונקציית step (מה- main). בכל קריאה לפונקציה זו ה- clock יעלה ב- 1.

### Step()

1. ביצוע fetch לפקודה הבאה.
2. אם יש פקודות ב- waitingList – ז"א יש פקודות שביצעו issue אך עדיין אינן מוכנות ל- execute, נטפל ברשימה זו על ידי הפונקציה handleWaitingList: פקודות שמוכנות לבצע execute נעביר ל- executeList.
3. נשלוף את הפקודה הבאה מתוך ה- instructions queue: אם היא לא null ואינה HALT נבצע לה issue. אם היא HALT, נגדיר את ה- issue\_cycle & execute\_cycle שלה כ- cycle הנוכחי ונשלוף אותה מתוך ה- instructions queue.
4. אם יש פקודות ב- executeList, נקרא לפונקציה execute.
5. אם יש פקודות ב- writeToCDBList, נקרא לפונקציה writeToCDB.
6. עבור כל הפקודות שביצעו issue במחזור שעון הנוכחי ומוכנות ל- execute, נעביר אותן ל- executeList, על מנת שיוכלו לבצע execute במחזור שעון הבא (הן יבצעו execute במידה והן יכולות להיכנס ליחידה הרלוונטית – בדיקה זו מתבצעת בפונקציה execute).
7. עבור כל היחידות שסיימו execute וצריכות להיכתב ל- CDB, נעביר אותן ל- writeToCDBList, כל מנת שיכתבו ל- CDB במחזור שעון הבא.
8. אם כל הרשימות ריקות (זאת אומרת ניתן לסיים את התוכנית ולצאת), נגדיר את ה- globalStatus כ- finished.

### Issue()

1. נשלוף את ההוראה הבאה מה- instructions queue.
2. נגדיר את ה- issue\_cycle של ההוראה כמחזור שעון הנוכחי.
3. בהתאם ל- opcode של ההוראה: אם יש לה באפר / תחנה פנויה, נקצה לה באפר / תחנה ונכניס את הערכים הרלוונטיים לתוך הבאפר / התחנה. אם מדובר בפעולת jump: נשנה את ה- pc בהתאם ונרוקן את ה- instructions queue. אם מדובר בפעולת branch: נבצע branch resolution.

### branchResolution()

1. אם הרגיסטר ב- SRC0 מעודכן, נשלוף את הערך מרגיסטר זה. אחרת, נחזור (הפעולה לא תישלף מתוך ה- instructions queue וכך במחזור שעון הבא ננסה לעשות לה שוב issue).
2. אם הרגיסטר ב- SRC1 מעודכן, נשלוף את הערך מרגיסטר זה. אחרת, נחזור (הפעולה לא תישלף מתוך ה- instructions queue וכך במחזור שעון הבא ננסה לעשות לה שוב issue).
3. 2 התנאים הקודמים מתקיימים, ולכן ניתן לבדוק האם אנחנו מבצעים branch או לא. על כן נגדיר את ה- execute\_cycle של ה- instruction במחזור שעון הנוכחי. אם צריכה להתבצע קפיצה, נשנה את ה- pc בהתאם ונרוקן את ה- instructions queue. אחרת, נשלוף את ההוראה מה- instructions queue ונמשיך.

### Execute()

1. נעבור על ה- executeList. אם ההוראה נכנסה לרשימה זו במחזור שעון הנוכחי, נגדיר את זמן תחילת ביצוע ה- execute על ידי זה שנבדוק כמה הוראות מחכות ליחידה הרלוונטית, מאחר והן מצונררות. לזמן זה נוסיף את ה- delay של היחידה הרלוונטית, וכך נקבע את זמן סיום ביצוע ה- execute.
2. אם מחזור השעון הנוכחי הוא המחזור שעון האחרון של ההוראה, נבצע את ה- execute כעת על ידי הפונקציה executeInstruction, הקוראת לפונקציה execute של היחידה הרלוונטית. בנוסף, נוציא את ההוראה מה- executeList, ונוסיף אותה ל- writeToCDBList (בפועל אנחנו מוסיפים אותה לרשימה זמנית, ובסיום המחזור שעון אנחנו מוסיפים ל- writeToCDBList, על מנת שהיא בפועל תכתוב ל- CDB במחזור שעון הבא).

### writeToCDB()

1. נעבור על ה- writeToCDBList ועבור כל הוראה (אם אינה הוראת store), נכתוב לכל הרגיסטרים, באפרים ותחנות המחכות ל- tag זה. בנוסף, נגדיר את ה- write2CDB\_cycle של ההוראה, ונשחרר את התחנה / באפר של ההוראה, על מנת שיוכל להיות זמין להוראות הבאות המחכות ל- issue.
2. ננקה את ה- writeToCDBList.

## פונקציות תומכות של המחלקה Tomasulo

* initializeBuffers – אתחול כל הבאפרים.
* initializeReservationStations – אתחול כל ה- reservation stations.
* initializeUnits – אתחול כל היחידות.
* handleWaitingList – טיפול בהוראות המחכות ל- execute: בדיקה האם הן מוכנות לבצע execute, כלומר אינן תלויות ב- tags מסויימים.
* fetchInstruction – ביצוע fetch של ההוראה הבאה.
* emptyInstructionsQueue – ניקוי ה- instructions queue (במקרים של jump / branch).
* setAluReservationStationValues – השמת כל ה- values / tags הרלוונטיים של התחנה של ההוראה שכעת מבצעת issue (עבור הוראות שנכנסו ל- ALU reservation station).
* setFloatReservationStationValues – השמת כל ה- values / tags הרלוונטיים של התחנה של ההוראה שכעת מבצעת issue (עבור הוראות שנכנסו ל- MUL / ADD reservation station – מסוג float).
* setLoadBufferValues - – השמת כל ה- values / tags הרלוונטיים של ה- load buffer של ההוראה שכעת מבצעת issue.
* setStoreBufferValues– השמת כל ה- values / tags הרלוונטיים של ה- store buffer של ההוראה שכעת מבצעת issue.
* getNextAvailableCycle – בשלב ה- execute: בדיקה מתי התחנה הבאה תוכל לקבל את ההוראה, בהתאם למספר ההוראות המחכות לתחנה.
* executeInstruction – קריאה לתחנה הרלוונטית של ההוראה וביצוע execute.
* updateUnit – עדכון ה- unit הרלוונטי של ההוראה (כמה הוראות מחכות ליחידה זו).
* getDelay – שליפת ההשהייה של התחנה הרלוונטית.
* printInstructions – הדפסת ההוראות (לצרכי debug).
* printRegistersValues – הדפסת ערכי הרגיסטרים (לצרכי debug).
* printReservationStations – הדפסת מצב ה- reservation stations / buffers (לצרכי debug).
* isFinished – מתן אינדיקציה בנוגע להאם התוכנית הסתיימה.