24.3.24

**ארכיטקטורה של מחשבים**

**פרויקט תכנות – Tomasulo**

יונתן ארז 208987073 עידו שני 203240353

**תוכן עניינים**

1. **הרצת קוד הפרויקט** – עמ׳ 3
2. **הנחות מימוש** – עמ׳ 4
3. **הלולאה המרכזית של הקוד** – עמ׳ 5
4. **מבני הנתונים** – עמ׳ 6
5. **וריפיקציה** – עמ׳ 11
   1. שיטת הוריפיקציה – כללי
   2. טבלת attributes ומקרי קצה
   3. פירוט הבדיקות שנעשות בכל אחד מהטסטים שהגשנו
6. **כלי עזר לדיבאג** – עמ׳ 16

**1 - הרצת קוד הפרויקט**

יצרנו קובץ אשר מבצע ביחד **קומפילציה והרצה** של הקוד עבור קבצי הקלט המתאימים - **run.sh**.

בדקנו את הקוד על מחשבי MAC, הן על intel x86 והן על ARM64 באמצעות קומפיילר GCC ( לא הייתה הנחיה מפורשת באיזה קומפיילר להשתמש, ולכן בחרנו בקומפיילר cross platform נפוץ ).

בקומפילציה באמצעות build יש לקמפל בנפרד את כל אחת מהתלויות ולבסוף את הקובץ המרכזי.

**2 – הנחות מימוש**

הנחנו את הנקודות הבאות:

1. **שחרור ה-FU במידה ומתבצעת ארביטרציה על ה-CDB ( אושר ע״י גדי ):**

מחזור ה-execute end תלוי אך ורק במחזור ה-execute start וה-delay של ה-FU המתאים. **שחרור ה-FU יתבצע בסוף מחזור ה-execute end.**

במידה וקיימת ארביטרציה על ה-CDB, אנו מניחים שתוצאת ה-FU של ההוראה שלא ״זכתה״ נשמרת בזיכרון ביניים ( לדוג׳, כזה שמשויך ל-reservation station ) – ה-FU לא יתקע וישוחרר למרות שמחזור ה-write CDB יתעכב.

1. **קובץ הקונפיגורציה:**
   1. את הפרמטרים בקובץ הקונפיגורציה: num reservation stations, num of functional units, delay ניתן **לייצג ע״י char** – כלומר מדובר בכמות ״קטנה״.
   2. **אף פרמטר לא מכיל 0.**

* gjgjgאח
* הvjחהחהל

**3 - הלולאה המרכזית של הקוד**

בחרנו לממש את הלולאה באופן הבא:

// runs the simulation till no pending instructions

void simulate(CPU\* cpu, SimArgs sim\_args)

{

FILE\* memin\_fp = fopen(sim\_args.memin, "r");

do

{

wrapper\_write\_cdb\_update\_rs\_qjk\_when\_needed(cpu);

write\_cdb\_update\_register\_array (cpu);

write\_cdb\_delete\_rs(cpu); // deleting the res. stations from prev. round. Immediately after that they would be cleaned

clean(cpu);

execute\_to\_write\_cdb(cpu, sim\_args.tracecdb);

issue\_to\_execute\_start(cpu);

issue(cpu);

if (!cpu->halt)

{

fetch(cpu, memin\_fp);

}

cpu->cycle++;

} while (cpu->inst\_state\_lst);

fclose(memin\_fp);

}

**הערה כללית:** לוגית, write CDB מסתיים בסוף מחזור ה-write CDB. ברמת המימוש, כתיבת הערכים ליעדים קורית בתחילת המחזור הבא ( וכמובן שאנו מטפלים בנפרד במקרה המיוחד של issue ו- write CDB באותו מחזור שעון.

**השלבים המרכזיים:**

1. עבור write CDB שקרה במחזור הקודם:
   1. עדכון Qj,k בתחנות ההמתנה, במידה ונדרש‎.
   2. כתיבת הערך למערך הרגיסטרים, במידה ונדרש.
   3. מחיקת תחנות ההמתנה אשר ערכן חושב ונכתב דרך ה CDB.
   4. ניקוי כלל הפקודות שהסתיימו מתור ההוראות.
2. העברת פקודות מ-execute done ל-write CDB במידת האפשר.
3. העברת פקודות משלב ה-issue לשלב execute במידת האפשר.
4. ביצוע issue לפקודות במידת האפשר.
5. בדיקת האם קיבלנו HALT, אם לא – ביצוע fetch במידת האפשר.
6. הגדלת מחזור השעון.

אם אין עוד פקודות בתור ההוראות נסיים את ביצוע התוכנית.

**4 - מבני נתונים**

typedef struct SimArgs {

char\* cfg;

char\* memin;

char\* regout;

char\* traceinst;

char\* tracecdb;

} SimArgs;

מבצע פיצול של הקלט למספר משתנים (מטעמי נוחות).

typedef struct CPU {

bool halt;

int cycle;

int pc;

InstStateNode\* inst\_state\_lst;

InstStateTrace\* inst\_state\_trace;

RegState reg\_state\_arr [REGISTERS\_AMOUNT];

LogicalUnit\* logical\_unit\_arr [LOGICAL\_UNIT\_TYPES];

CdbState cdb\_state\_arr [LOGICAL\_UNIT\_TYPES];

} CPU;

היחידה המנהלת את המעבד.

דגל האם התקבל HALT.

מונה המציין את מחזור השעון הנוכחי.

מונה פקודה pc.

רשימה מקושרת המחזיקה את הפקודות שנקראו מהזיכרון.

רשימה מקושרת המחזיקה את הפקודות אשר בוצעו.

מערך בגודל 16 המייצג את טבלת הרגיסטרים.

מערך בגודל 3 המייצג את היחידות הלוגיות.

מערך בגודל 3 המייצג את יחידת CDB.

typedef struct InstStateNode {

InstState\* inst\_state;

struct InstStateNode\* next;

} InstStateNode;

רשימה מקושרת המייצגת את הפקודות המצויות אשר נקראו מהזיכרון.

מחזיקה את תיאור הפקודה.

מצביע לאיבר הבא.

typedef struct InstState {

Inst inst;

int pc;

int cycle\_issued;

Tag res\_sta\_tag;

int cycle\_execute\_start;

int cycle\_execute\_end;

int cycle\_write\_cdb;

int cycle\_fetched;

} InstState;

הפקודה המתקבלת.

מונה פקודה.

מחזור השעון בו הפקודה התקבלה.

סימון יחידת המתנה תואמת.

מחזור השעון בו התחיל החישוב.

מחזור השעון בו החישוב יסתיים.

מחזור השעון בו ייכתב החישוב.

מחזור השעון בו קראנו את הפקודה מהזיכרון, (DEBUG).

typedef struct Inst {

char opcode;

char dst;

char src0;

char src1;

} Inst;

פיצול הפקודה לצורך חישוב והשמה.

אופרטור אריתמטי.

רגיסטר יעד.

רגיסטר מקור ראשון.

רגיסטר מקור שני.

typedef struct Tag {

int type;

int res\_sta\_idx;

} Tag;

סימון גורף של תחנת ההמתנה הרלוונטית.

אינדקס סוג.

אינדקס לתחנה מתוך מערך התחנות של היחידה הלוגית.

typedef struct InstStateTrace {

InstState inst\_state;

struct InstStateTrace\* next;

} InstStateTrace;

רשימה מקושרת המחזיקה את כלל הפקודות מתחילת הריצה.

מחזיקה שדה של תיאור הפקודה.

מצביע לאיבר הבא.

typedef struct RegState {

float v;

Tag q;

} RegState;

ייצוג של מצב רגיסטר יחיד ממערך הרגיסטרים.

שדה ערך נוכחי.

שדה סימון לתחנה ההמתנה בו נמצא הערך אליו מחכים.

typedef struct LogicalUnit {

int nr\_fus;

int nr\_avail\_fus;

int fu\_delay;

int nr\_res\_stas;

int nr\_avail\_res\_stas;

ResSta\* res\_sta\_arr;

} LogicalUnit;

ייצוג של יחידה לוגית.

מונה מספר יחידות החישוב מסוג זה.

מונה של מספר היחידות החופשיות.

מונה של זמן ההמתנה ליחידה לוגית מסוג זה.

מונה של מספר יחידות ההמתנה הקיימות.

מונה של מספר יחידות ההמתנה החופשיות.

מצביע למערך יחידות ההמתנה.

typedef struct ResSta {

Tag tag;

bool busy;

float vj, vk;

Tag qj, qk;

} ResSta;

ייצוג של יחידת המתנה.

סימון של היחידה הנוכחית.

דגל האם היחידה בשימוש.

ערכי קלט.

סימון של יחידות ההמתנה בהן נמצאי ערכי הקלט אליהם מחכים.

typedef struct CdbState {

bool cdb\_used;

float cdb\_value;

Tag res\_sta\_tag;

int dst\_reg;

bool update\_reg\_file;

} CdbState;

ייצוג של יחידת CDB.

דגל האם היחידה בשימוש.

ערך אותו נרצה לכתוב.

סימון ליחידת ההמתנה ממנה לקחנו את הערך.

אינדקס מספר הרגסיטר אותו נעדכן במערך הרגיסטרים.

דגל האם לעדכן את הרגיסטר במערך הרגיסטרים.

**5 – וריפיקציה**

**5.1. שיטת הוריפיקציה – הסבר כללי**

ראשית, וידאנו ש**קובץ הבדיקה של גדי** עבר בהצלחה.

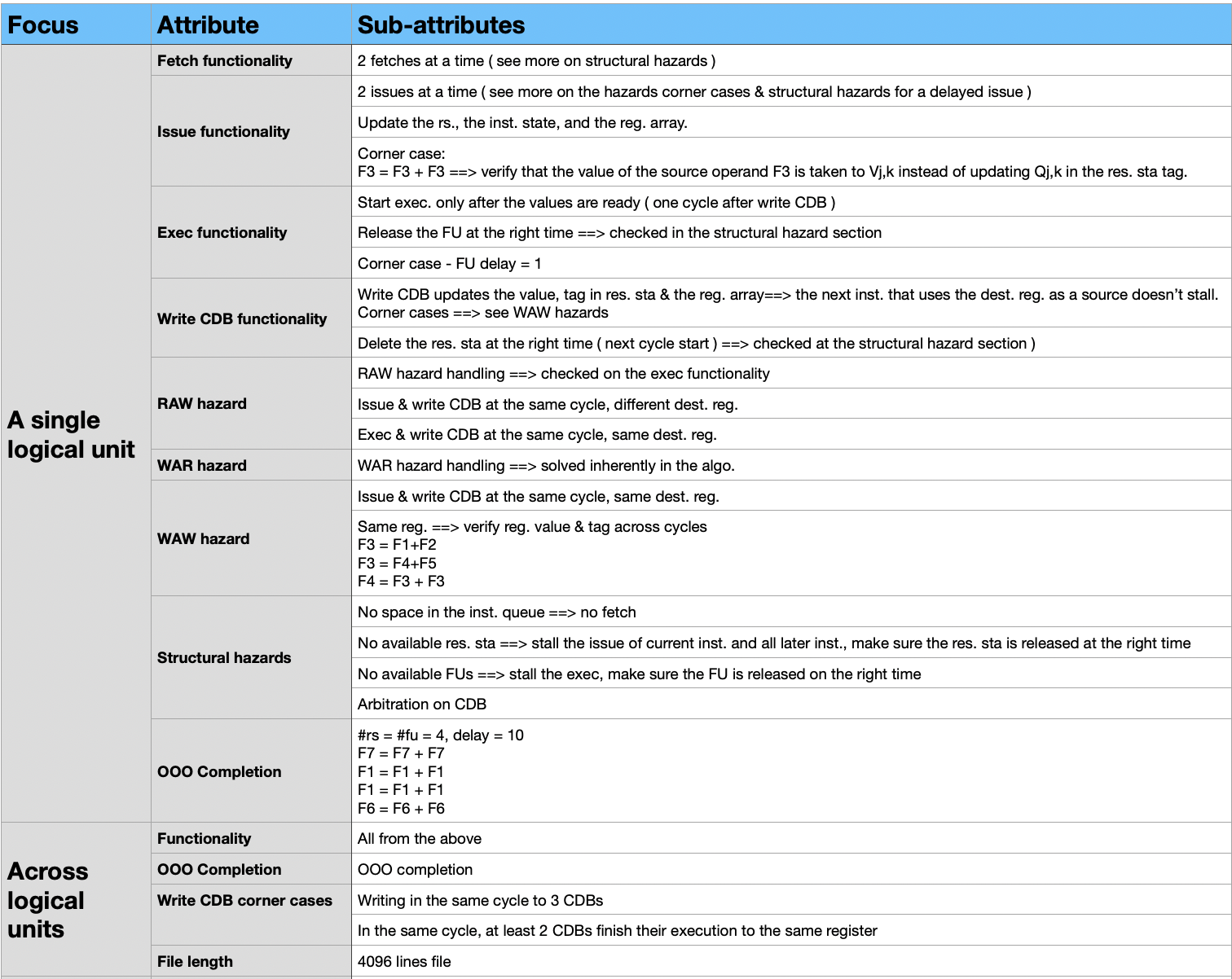
לאחר מכן, עברנו שוב על האלגוריתם, וכתבנו בטבלה המפורטת מטה מהם **ה-attributes ומקרי הקצה** שאנו רוצים לוודא. בנוסף, עברנו על **מספר רב של מבחנים משנים קודמות** – אומנם לא ניתן להשתמש בשאלות מהמבחן “as is” ( היחידות הפונקציונליות במבחנים הן pipelined וה-issue הוא עד פקודה אחת במחזור שעון ) – אבל וידאנו עד כמה שניתן שלא פסחנו על מקרה קצה מסוים שבא לידי ביטוי בפתרונות.

מיפינו את ה-attributes לטסטים שכתבנו בעצמנו - ראשית כל יחידה פונקציונלית בפני עצמה על כל שלביה, ולאחר מכן השילוב של שלושתן.

**Checkers**:

* זמני הביצוע של כל אחד מהשלבים: את ה-checkers כתבנו ידנית – מהם הערכים שצריכים להתקבל בסיום האלגוריתם. באופן כללי ניתן ״לסמוך״ על Tomasulo שיודע לטפל ב-hazards מסוג WAW ו-WAR, ולבדוק במפורש את התלויות עבור RAW.
* ערכי הרגיסטרים: בנינו סקריפט פייתון שמבצע חישוב של הערכים בסיום בצורה סריאלית, עם ״זמן רב״ בין הוראה להוראה.

**5.2. טבלת attributes ומקרי קצה** ( ב-4.3 נסביר כיצד הם ממופים לטסטים שהגשנו )



**5.3. טסט לבדיקה מספר 1**

**מטרת הטסט:** לבדוק כל יחידה פונקציונלית בפני עצמה ( החלפנו אח״כ את התפקידים בין היחידות ). בצידי כל הוראה מופיע מה רצינו לבדוק

**מקרא:**

// 0, 1, 2, 3, 4 🡺 fetch cycle, issue cycle, exec start cycle, exec end cycle, write CDB cycle

**פירוט הטסט בצורה ״מפוענחת״:**

**// Cfg. values:**

**// Add: #rs = 5, #fu = 4, delay = 2**

**// mul: #rs = 5, #fu = 4, delay = 2**

**// DIV: #rs = 2, # fu= 1, delay = 3**

**// ADD**

**F2 = F2 + F2 // 0, 1, 2, 3, 4**

**F1 = F1 + F1 // 0, 1, 2, 3, 5 // Add CDB arbitration, order of operations within issue ( read value ==> write tag on reg. file )**

**F1 = F1 + F1 // 1, 2, 6, 7, 8 // RAW F1 + WAW F1**

**F3 = F3 + F3 // 1, 2, 3, 4, 6 // In order issue, OOO execution & completion**

**F3 = F1 + F1 // 2, 3, 9, 10, 11 // WAW F3 + RAW F1**

**F4 = F1 + F1 // 2, 5, 9, 10, 12 // Structural on res. sta ==> delayed issue; RAW corner case: issue & write CDP on the same cycle (F1); structural on ADD CDB arbitration**

**F6 = F2 + F5 // 3, 6, 7, 8, 9 // Issue is stalled due to prev. inst., OOO execution**

**F5 = F1 + F1 // 3, 7, 9, 10, 13 // structural on ADD CDB arbitration: 3 at a time!**

**// DIV**

**F7 = F14 / F7 // 4, 7, 8, 10, 11**

**F15 = F14 / F15 // 4, 8, 11, 13, 14 // Structural on DIV FUs**

**F15 = F15 / F15 // 5, 8, 15, 17, 18 // RAW F15**

**// MULT - We've taken the same ADD inst. and changed the register names ( a disjoint set of registers ).**

**// We've made sure that the issue "initial conditions" are the same, hence the issue to write cdb cycles' timing is the same, w/ an offset of 9.**

**// By this test, we're making sure that the above corner cases work also on another type of commands**

**F8 = F8 \* F8 // 5**

**F9 = F9 \* F9 // 6**

**F9 = F9 \* F9 // 6**

**F10 = F10 \* F10 // 7**

**F10 = F9 \* F9 // 7**

**F11 = F9 \* F9 // 8**

**F13 = F8 \* F12 // 9**

**F12 = F9 \* F9 // 9**

**HALT**

**5.4. טסט לבדיקה מספר 2**

**מטרת הטסט:** לבדוק את נכונות האלגוריתם עבור שילוב של היחידות הפונקציונליות ( בפרט, OOO completion, כתיבה למספר CDBs באותו מחזור ולאותו רגיסטר, המקרים ה״מיוחדים״ של האלגוריתם, וכו׳ ). בצידי כל הוראה מופיע מה רצינו לבדוק.

**פירוט הטסט בצורה ״מפוענחת״:**

**// Cfg. values:**

**// Add #res. sta = 3**

**// Add latency = 1**

**// Add #fus = 1**

**// Mul #res. sta = 2**

**// Mul latency = 2**

**// Mull #FUs = 2**

**// DIV #res. sta = 2**

**// DIV latency – 4**

**// DIV #FUs = 1**

**// The inst.:**

**F4 = F2 + F2 // 0,1, 2, 2, 3**

**F5 = F1 / F2 // 0, 1, 2, 5, 6 // Part of WAW on F5, w/ 3 CDBs writing to the same reg.**

**F15 = F14 \* F13 // 1, 2, 3, 4, 5 // Writing in 2 CDBs in the same cycle to different regs.**

**F5 = F4 \* F2 // 1, 2, 4, 5, 6 // RAW for already issued cmd on F4 + WAW F5 ==> ex. start can happen only in cycle 4 and not in cycle 3; part of WAW w/ 3 CDBs writing to the same reg.**

**F4 = F1 + F4 // 2, 3, 4, 4, 5 // RAW “special case”, same dest reg. val + corner case of dst\_reg = source\_reg ==> taking for the issue the value at the beginning of the clk cycle**

**F5 = F1 + F2 // 2, 3, 5, 5, 6 // arbitration on FU ==> exec. start can start only after cycle 5; WAW w/ 3 CDBs writing to the same reg.**

**F7 = F5 \* F10 // 3, 6, 7, 8, 9 // structural on #MUL res. sta**

**F1 = F1 + F1 // 4, 6, 7, 7, 8 // issue delay, corener case of usage of the same register**

**F15 = F1 / F15 // 4, 7, 9, 12, 13 // RAW corner case: write CDB and exec on the same cycle**

**halt**

**5.5 טסט לבדיקה מספר 3**

**מטרת הטסט:** לבדוק את נכונות ה-fetch – עד 2 הוראות בו״ז במחזור שעון, ואת ה-structural hazard – עד 16 הוראות בו״ז בחלון ההוראות.

**פירוט הטסט בצורה ״מפוענחת״:**

**// Cfg. values:**

**// Add #res. sta = 17**

**// Add latency = 1**

**// Add #fus = 17**

**// Mul #res. sta = 17**

**// Mul latency = 5**

**// Mull #FUs = 1**

**// DIV #res. sta = 17**

**// DIV latency = 20**

**// DIV #FUs = 20**

**// The inst.:**

**F0 = F0 / F1**

**F2 = F2 / F1**

**F2 = F2 / F1**

**F3 = F3 / F1**

**F4 = F4 / F1**

**F5 = F5 / F1**

**F6 = F6 / F1**

**F7 = F7 / F1**

**F8 = F8 / F1**

**F9 = F9 / F1**

**F10 = F10 / F1**

**F11 = F11 / F1**

**F12 = F12 / F1**

**F13 = F13 / F1**

**F14 = F14 / F1**

**F15 = F15 / F1**

**F1 = F1 + F1**

**F10 = F10 \* F10**

**Halt**

**// Elaboration:**

**// Structural hazard on inst. queue: pc 17 is delayed until pc0 is completed.**

**// PC18 is not released on the same cycle, but only on the next one, when PC1 is completed.**

**// PC0, PC1 ==> Arbitration on the same DIV CDB cycle**

**// RAW for PC2,3**

**6 – כלי עזר לדיבאג**

בנינו פונקציות עזר אשר מסייעות לבדיקת הקוד. פונקציית ה-wrapper - print status עוברת על הסטטוס הנוכחי של טבלת הפקודות, טבלת הרגיסטרים וסטטוס היחידות הלוגיות ומדפיסה אותם.

void print\_status(CPU\* cpu){

printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ CYCLE #%d ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n", cpu->cycle);

printf("cpu->halt = %s\n\n", cpu->halt?"true":"false");

print\_instructions\_state\_status(cpu->inst\_state\_lst);

print\_registers\_status(cpu->reg\_state\_arr);

print\_logical\_units\_status(cpu->logical\_unit\_arr);

}

בנוסף, יצרנו קובץ memin\_generator.py אשר הופך פקודות אשר נכתבות בפורמט נוח ומייצג opcode. כאמור, חישבנו בעזרת קובץ זה את התוצאות הרצויות של ערכי הרגיסטרים בסיום הריצה על ידי חישוב סריאלי.