**Simulation Flow**

אנו 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);

}

1. בדיקה האם אחת מתחנות ההמתנה מחכה לערך מחושב הקיים ב CDB.
2. כתיבת הערכים הקיימים ב CDB למערך הרגיסטרים.
3. איפוס תחנות ההמתנה אשר ערכן חושב ונכתב דרך ה CDB.
4. ניקוי כלל הפקודות אשר בוצעו ויכולות להימחק.
5. העברת כלל הפקודות משלב ISSUE לשלב EXECUTE.
6. ביצוע ISSUE לכלל הפקודות אשר נקראו.
7. בדיקת האם קיבלנו HALT, אם לא נקרא פקודה מהזיכרון.
8. נמשיך לסיבוב הבא.
9. אם אין עוד פקודות ממתינות נסיים.

**Data Structures**

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.

**Testing**

בנינו פונקציות עזר אשר מסייעות לבדיקת הקוד.

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);

}

פונקציה זו עוברת על הסטטוס הנוכחי של טבלת הפקודות, טבלת הרגיסטרים וסטטוס היחידות הלוגיות.

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

**Compilation & Exectution**

יצרנו קובץ אשר מבצע קומפילציה והרצה של הקוד עבור קבצי הקלט התואמים - run.sh

בקובץ יש את פקודת הקומפילציה ואת ההרצה הרלוונטית של כל אחד מהשלבים.

אנו מניחים כי לא קיים קובץ מעקב אחר יחידת CDB ואנו מנקים אותו טרם תחילת ריצת התכנית.