ת"ב 3 – ניתוח תלויות מידע

קצר ולעניין

מיקרו-ארכיטקטורה המיישמת את עקרונות Out-Of-Order-Execution שואפת למעשה לנצל את המקביליות הטמונה ברצף הפקודות לביצוע. **אי** תלות של האופרנדים של פקודות בתוצאה של פקודות מוקדמות יותר בתכנית מאפשרת להריץ מספר פקודות במקביל, ביחידות פונקציונליות שונות. יחסי התלויות בין פקודות בתכנית מוגדר כתלויות מידע. בניתוח תלויות מידע בתכנית אנו מתייחסים ל**תלויות האמיתיות (RAW) בלבד**, מכיוון שתלויות לא אמיתיות (false dependencies) ניתנות לפיתרון בעזרת מנגנונים מתאימים, כמו register renaming.

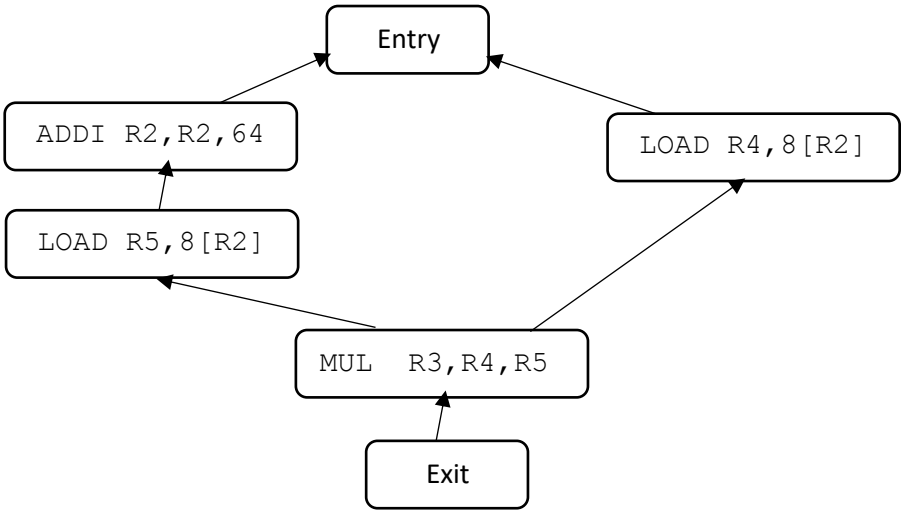
באמצעות ניתוח תלויות המידע ניתן לחשב את המקביליות התאורטית שקיימת בתכנית. מידע זה יכול לשמש לתכנון מתאים של המיקרו-ארכיטקטורה ולהקצאת משאבים במעבד באופן אופטימלי המאזן בין שיפור הביצועים באמצעות מיקבול לבין כמות משאבי החומרה.

בתרגיל זה תממשו מנתח תלויות מידע בין פקודות של תכנית נתונה. כרגיל, מוגדר ממשק למימוש שלכם. הממשק מקבל את מאפייני המעבד ועקבות ריצה של תכנית ולאחר מכן מאפשר ביצוע שאילתות לגבי תלויות המידע של פקודות בתכנית, למשל: באיזה פקודות תלויה פקודה מסוימת.

אלגוריתם ניתוח תלויות המידע

תלויות מידע בין פקודות ניתנות לתיאור כגרף מכוון, שבו כל צומת היא פקודה וממנה יוצאות קשתות לפקודות שהיא תלויה בהן. הפקודות בתחילת רצף הפקודות (שלא תלויות בפקודות בתכנית הנתונה) תצבענה לצומת מיוחדת שתקרא Entry - צומת זו מייצגת את תחילת ביצוע התכנית והקלט שמקבלת (מידע שמסופק ממקור חיצוני, למשל, פרמטרים לפונקציה). כמו-כן, תוגדר צומת מיוחדת בשם Exit שתלויה בכל הפקודות שאין פקודה אחרת בתכנית שתלויה בהן, כך שכאשר מתקיימות כל התלויות של Exit נדע כי התכנית הסתיימה. לדוגמה, עבור התכנית:

```
LOAD R4,8[R2]
ADDI R2,R2,64
LOAD R5,8[R2]
MUL  R3,R4,R5
```



לכל פקודה במעבד נתון משך זמן ביצוע בהתאם למאפייני היחידה הפונקציונלית שמבצעת אותן. למשל, בדוגמה לעיל נגדיר ADDI - 1 מ"ש, MUL - 2 מ"ש, LOAD - 5 מ"ש. נוכל לסכם את משכי הביצוע של הפקודות במסלול מפקודה בגרף לצומת Entry . המסלול בעל משך הזמן המקסימלי מייצג את "עומק" התלויות של פקודה – כלומר, הזמן המינימלי עד לתחילת ביצוע הפקודה, בהנחה שאין מגבלה הנובעת מזמינות משאבי מערכת - Structural Hazard. עומק התלויות של התכנית הינו המסלול הארוך ביותר (במחזורי שעון) מ-Entry ל-Exit – שהוא גם החסם המינימלי למשך ביצוע התכנית.

בדוגמה הנ"ל, בהתאם להגדרת משכי הפקודות הנתונה, עומק התלויות בתכנית (כלומר, עומק Exit) הוא 8, על פי המסלול הארוך (השמאלי בגרף) מבחינת משכי ביצוע הפקודות במסלול.

שימו-לב כי בגלל משכי זמן שונים של ביצוע ביחידות פונקציונליות שונות עבור פקודות שונות (למשל, יחידת חיבור/חיסור לעומת יחידת חילוק), ייתכן מסלול בעל יותר פקודות - כלומר, יותר צמתים במסלול - שדורש פחות זמן ביצוע ממסלול אחר בעל פחות פקודות. אורך המסלול הקובע לחישוב עומק התלות מתייחס לסכום משך ביצוע הפקודות במסלול **ולא** למספר הצמתים (פקודות) במסלול.

אז מה תכל'ס צריכים לעשות?

אנחנו מספקים לכם קובץ dflow_main.c שקורא שני קבצי קלט: האחד מספק את משכי הביצוע של כל סוג פקודה (opcode) במעבד נתון, והשני מספק רשימת פקודות מעקבות ריצה (trace) של תכנית, כמפורט בפרק הבא. ה-main קורא לפונקציית המנתח שלכם על מנת שינתח את תלויות המידע בתכנית הנתונה (יבנה גרף תלויות, וכו'). לאחר מכן ה-main קורא לפונקציות שאילתה לגבי מאפייני תלויות המידע בתכנית.

עליכם לממש את מנתח התלויות בקובץ dflow_calc.c או dflow_calc.cpp . על המנתח לממש את הפונקציות המוגדרות בקובץ dflow_calc.h:

1. פונקציית אתחול המנתח עבור תכנית מסוימת. הפונקציה אמורה לנתח את תלויות המידע בתכנית ולהחזיר "ידית" (handle) למבנה הנתונים שלכם שמחזיק את תוצאות הניתוח עבור התכנית, לשימוש בשאר הפונקציות של הממשק.

```
ProgCtx analyzeProg(const unsigned int opsLatency[],
                    const InstInfo progTrace[],
                    unsigned int numOfInsts);
```

2. פונקציה לשחרור משאבים שהוקצו בפונקציית האתחול. לשימוש **לאחר** סיום השימוש בפונקציות השאילתה עבור תכנית מסוימת.

```
void freeProgCtx(ProgCtx ctx);
```

3. פונקציה לקבלת "עומק" התלויות של פקודה בריצת התכנית. הפונקציה אמורה להחזיר את אורך מסלול התלויות הארוך ביותר (במחזורי שעון) מ-Entry עד לתחילת הפקודה (כלומר, לא כולל משך ביצוע הפקודה עצמה). עבור פקודות שתלויות רק ב-Entry (לא תלויות בפקודות קודמות בריצה), הפונקציה תחזיר 0.

```
int getInstDepth(ProgCtx ctx, unsigned int theInst);
```

4. פונקציה לקבלת התלויות הישירות של פקודה בריצת התכנית. לכל פקודה יכולות להיות עד שתי תלויות. תלות מתוארת באמצעות המספר הסידורי של הפקודה שמפיקה את הנתון שתלויים בו, החל מאפס עבור הפקודה הראשונה בעקבות התכנית. עבור תלות לא קיימת (כלומר, תלות ב-Entry) יוחזר הערך 1- עבור המקור המתאים. שימו לב שהמספרים הסידוריים של התלויות מוחזרים למשתנים בכתובות הנתונות בפרמטרים `src1DepInst` ו-`src2DepInst`, וערך החזרה של הפונקציה משמש רק להחזרת קוד הצלחה או כישלון של הפונקציה. ראו פרטים נוספים בקובץ dflow_calc.h.

```
int getInstDeps(ProgCtx ctx, unsigned int theInst,
                int *src1DepInst, int *src2DepInst);
```

5. פונקציה לקבלת "עומק" התלויות של כל ריצת התכנית (מ-Entry ל-Exit) במחזורי שעון.

```
int getProgDepth(ProgCtx ctx);
```

הרצת המנתח

קובץ ה-makefile המסופק לכם מקמפל את קובץ המימוש שלכם ואת קובץ ה-main המסופק ומלנקג' אותם לקובץ ריצה בשם dflow_calc . הרצת המנתח ללא פרמטרים תדפיס הסבר לגבי אופן השימוש והפרמטרים לתכנית.

קובץ הריצה דורש שני פרמטרים: הראשון הוא שם קובץ הגדרת ההשהיות של הפקודות, והשני הוא שם קובץ עקבות הריצה. בנוסף, ניתן להוסיף מספר כלשהו (0 או יותר) של "שאילתות" בשורת הפקודה. כל שאילתה מתחילה באות ובצמוד אליה מספר פקודה שהיא מטרת השאילתה. האות p תשאל לגבי עומק התלויות של הפקודה הנתונה. האות d תשאל לגבי מספרי הפקודות שהפקודה הנתונה תלויה בהם. לדוגמה:

```
./dflow_calc opcode1.dat prog5.in p3 d5 p0 p12
```

דוגמה זו תטען את נתוני ההשהיות של הפקודות מהקובץ opcode1.dat ואת עקבות ריצת התכנית מ-prog5.in. היא תדפיס את העומק הכולל של התכנית, ובנוסף את העומק של פקודה 3, התלויות של פקודה 5, את העומק של פקודה 0 (הראשונה בעקבות הריצה), ואת העומק של פקודה 12.

מבנה קבצי הקלט

עם חומרי התרגיל מסופקות מספר דוגמאות לקבצי הקלט וכן פלטי ריצה מתאימים. להלן תיאור מבנה הקבצים, על מנת לאפשר לכם להכין קבצי קלט נוספים לבדיקה.

קובץ השהיות הפקודות

קובץ ההשהיות מכיל שורה עבור כל סוג פקודה - opcode - שקיים במכונה, החל מ-opcode מספר 0. כל שורה מכילה מספר דצימלי יחיד שמגדיר את ההשהיה במחזורי שעון של ה-opcode המתאים למספר הסידורי של השורה.

קובץ עקבות ריצת התכנית

קובץ עקבות ריצת התכנית מכיל בכל שורה תאור של פקודה מריצת התכנית, על פי סדר הביצוע (commit) הארכיטקטוני. בכל שורה בקובץ יופיעו ארבעה מספרים בסדר קבוע:

1. קוד* (opcode) הפקודה
2. אינדקס** רגיסטר היעד
3. אינדקס** רגיסטר מקור 1
4. אינדקס** רגיסטר מקור 2

* קוד פקודה הוא המספר הסידורי של פקודה בקובץ ההשהיות הנ"ל.

** האינדקסים של הרגיסטרים מתייחסים לרגיסטרים הארכיטקטוניים.

הנחות נוספות:

- הניחו כי בארכיטקטורת היעד של התכנית המנותחת יש לכל היותר 32 רגיסטרים (ממוספרים 0 עד 31). אין צורך לטפל ברגיסטר 0 או בכל רגיסטר אחר באופן מיוחד (כלומר, כל השמה לכל רגיסטר משנה את תוכנו).
- במסגרת תרגיל זה אנו מתעלמים מתלויות מידע שנובעות מכתיבות/קריאות לזיכרון הראשי. כל התלויות מתייחסות לתלויות בין רגיסטרים בלבד.

דרישות ההגשה

הגשה אלקטרונית בלבד באתר הקורס ("מודל") מחשבונו של אחד הסטודנטים.

מועד ההגשה: עד יום א' 04.06.2017 בשעה 23:55.

עליכם להגיש קובץ ארכיב מסוג* tar.gz. בשם hw3_*ID1_ID2*.tar.gz כאשר ID1 ו-ID2 הם מספרי ת.ז. של המגישים.
לדוגמה: hw3_012345678_987654321.tar.gz . הארכיב יכיל קובץ **יחיד**:

- קוד המקור של המנתח שלכם: dflow_calc.c או dflow_calc.cpp .
  קוד המקור חייב להכיל תיעוד פנימי במידה סבירה על מנת להבינו.

* ראו דוגמאות לאופן יצירת קובץ tar.gz. בתרגילי הבית הקודמים.

דגשים להגשה:

1. המימוש שלכם **חייב** להתקמפל בהצלחה ולרוץ במכונה הוירטואלית שמסופקת לכם באתר הקורס. זוהי סביבת הבדיקה המחייבת לתרגילי הבית. יש לוודא בניה מוצלחת במכונה הוירטואלית באמצעות קובץ ה-makefile שמסופק לכם וקבצי העזר המקוריים. **קוד שלא יתקמפל יגרור ציון 0!**
2. מכיוון שאתם מגישים רק את קובץ המימוש, אסור לכם להסתמך על שינויים מקומיים שאתם עושים בקבצי מקור המסופקים לכם, או על שינויים שאתם עושים ב-makefile. עליכם לבדוק את המימוש שלכם (הקובץ היחיד שכלול בהגשה) עם הקבצים המקוריים שמסופקים לכם (מלבד קובץ השלד למימוש dflow_calc.c).
3. שימו-לב לסנקציות במקרה איחור כמפורט באתר הקורס. לא מומלץ לאחר את מועד ההגשה שהוגדר לתרגיל. בנסיבות מיוחדות יש לקבל אישור לאיחור **מראש** מהמתרגל האחראי.
4. מניסיונם של סטודנטים אחרים: הקפידו לוודא שהקובץ שהעלתם ל"מודל" הוא אכן הגרסה שהתכוונתם להגיש, מסוג קובץ הארכיב הנכון ועם התוכן המתאים. לא יתקבלו הגשות נוספות לאחר מועד ההגשה שנקבע בטענות כמו "משום מה הקובץ במודל לא עדכני ויש לנו גרסה עדכנית יותר שלא נקלטה".

בהצלחה!