



ת"ב 3 – ניתוח תלויות מידע

קצר ולעניין

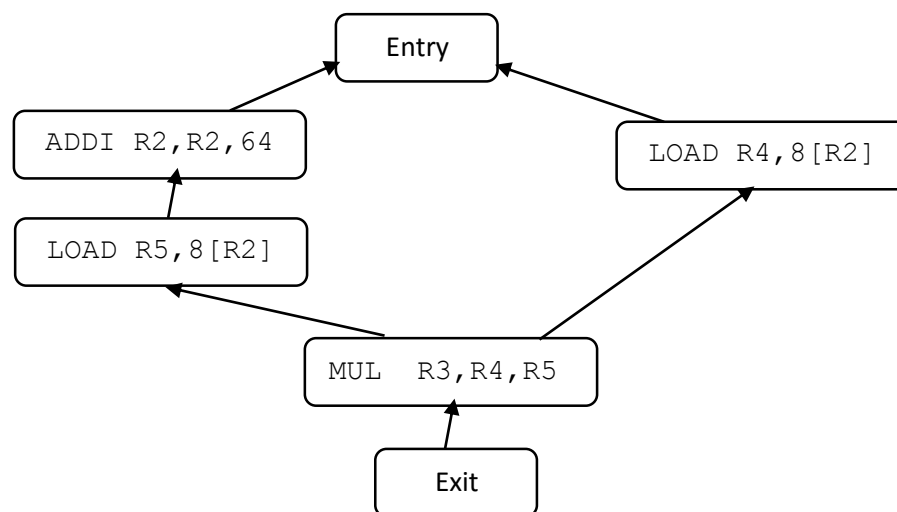
מיקרו-ארכיטקטורה המיישמת את עקרונות Out-Of-Order-Execution שואפת למעשה לנצל את המקביליות הטמונה ברצף הפקודות לביצוע. אי תלות של האופרנדים של פקודות בתוצאה של פקודות מוקדמות יותר בתכנית מאפשרת להריץ מספר פקודות במקביל, ביחידות פונקציונליות שונות. יחסי התלויות בין פקודות בתכנית מוגדר כתלויות מידע. באמצעות ניתוח תלויות המידע ניתן לחשב את המקביליות התאורטית שקיימת בתכנית. מידע זה יכול לשמש לתכנון מתאים של המיקרו-ארכיטקטורה ולהקצאת משאבים במעבד על מנת לשפר את הביצועים באמצעות מיקבול.

בתרגיל זה תממשו מנתח תלויות מידע בין פקודות של תכנית נתונה. כרגיל, מוגדר ממשק למימוש שלכם. הממשק מקבל את התכנית לניתוח 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 ידוע כי התכנית הסתיימה. לדוגמה, עבור התכנית:

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



אורך המסלול הארוך ביותר מפקודה בגרף לצומת Entry מייצג את "עומק" התלויות של פקודה. עומק התלויות של הצומת Exit מתאר את אורך המסלול הקריטי בתכנית שמייצג גם את החסם התאורטי להאצת התכנית באמצעות מיקבול (במקרה האופטימלי בו כל פקודה נמשכת מחזור שרון אחד ואין מגבלה הנובעת מזמינות משאבי מערכת - Structural Hazard וכו').


בנוסף לקשתות מה"צרכנים" (consumers) ל"יצרנים" (producers) של מידע, ניתן גם ליצור קשתות מה"יצרנים" ל"צרכנים". קשתות אלו יכולות לעזור להבין מיהן הפקודות הקריטיות – כלומר, כאלו שאופטימיזציה שלהן יכולה לשפר את ביצועי התכנית בצורה משמעותית, כי הן מהוות צוואר בקבוק למספר רב של פקודות.



אז מה תכל'ס צריכים לעשות?

אנחנו מספקים לכם קובץ `dflow_main.c` שקורא קובץ קלט המתאר 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 "ידיית" (`handle`) למבנה הנתונים שלכם שמחזיק את תוצאות הניתוח עבור התכנית, לשימוש בשאר הפונקציות של הממשק. 

```
ProgCtx analyzeProg(InstInfo prog[], 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 בתכנית. הפונקציה אמורה להחזיר את אורך מסלול התלויות הארוך ביותר מאותה פקודה ל-`Entry`, לא כולל `Entry` (עבור פקודות שתלויות רק ב-`Entry`).

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

4. פונקציה לקבלת התלויות הישירות של פקודה בתכנית. לכל פקודה יכולות להיות עד שתי תלויות. תלות מתוארת באמצעות המספר הסידורי של הפקודה שבה תלויים. עבור תלות לא קיימת (כלומר, תלות ב-`Entry`) יוחזר הערך -1.

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



5. פונקציה לקבלת הצרכנים של התוצאה של פקודה. מכיוון שמספר הצרכנים לא חסום, הפונקציה מקבלת מהקורא מערך עם גודל נתון. הפונקציה תמלא כניסות במערך שניתן בכמות מתאימה למספר הצרכנים בפועל ותחזיר את מספר הצרכנים. אם גודל המערך שניתן קטן מדי, ייכתבו רק מספר צרכנים כמקום שסופק (`numUsersIn` הראשונים, לפי סדר הפקודות בתכנית) אבל ערך ההחזרה עדיין יציין את המספר הכולל של צרכנים לפקודה (כדי שהקורא ידע מה גודל המערך שנדרש כדי לקבל תשובה מלאה).

```
int getInstUsers(ProgCtx ctx, unsigned int theInst,  
unsigned int numUsersIn, unsigned int *users);
```

* צומת `Exit` אינה נמנית על הצרכנים של פקודה.

מבנה קובץ הקלט

קובץ הקלט יכיל את רצף הפקודות של התכנית. בכל שורה בקובץ יופיעו שלושה מספרים המציינים אינדקסים של רגיסטרים ארכיטקטוניים בסדר קבוע:

1. רגיסטר היעד
2. רגיסטר מקור 1
3. רגיסטר מקור 2

לא יציין בקובץ סוג הפקודה. לדוגמה, אם בתכנית היתה הפקודה `ADD R7, R2, R23` בקובץ ירשם: 7 2 23

מסופק לכם עם חומרי התרגיל קובץ קלט לדוגמה.



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).
- הניחו כי אין פקודות סיעוף בתכנית המנותחת, כלומר, מקור של פקודה תלוי בפקודה האחרונה לפנייה בתכנית שכתבה לאותו רגיסטר ארכיטקטוני שהוא מקור בפקודה.
- בתרגיל זה אנו מתעלמים מתלויות מידע שנבעות מכתובות/קריאות לזיכרון הראשי. כל התלויות מתייחסות לרגיסטרים בלבד.
- כרגיל, ה-main הנתון הוא רק דוגמה ובבדיקה ייתכן ונקרא לפונקציות המנתח שלכם באופן שונה, למשל, שאליות שונות או ניתוח של מספר תכניות קלט.

דרישות ההגשה

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

מועד ההגשה: עד יום ג' 24.05.2016 בשעה 23:55

עליכם להגיש קובץ ZIP בשם `hw3_ID1_ID2.zip` כאשר ID1 ו-ID2 הם מספרי ת.ז. של המגישים. לדוגמה:
`hw3_012345678_987654321.zip`. ה-ZIP 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.

דגשים להגשה:

1. המימוש שלכם **חייב** להתקמפל בהצלחה ולרוץ בסביבת השרת הטכניוני T2 (או TX). יש להשתמש בקומפיילר G++/GCC ברירת המחדל בשרת ללא שום דגלים מיוחדים מעבר למה שמשתמשים ב-makefile המסופק לכם.
2. שימו-לב לסנקציות במקרה איחור כמפורט באתר הקורס. מאוד לא מומלץ לאחר את מועד ההגשה שהוגדר לתרגיל. בנסיבות מיוחדות יש לקבל אישור **מראש** מהמתרגל האחראי לאיחור.
3. מניסיונם של סטודנטים אחרים: הקפידו לוודא שהקובץ שהעלתם ל"מודל" הוא אכן הגרסה שהתכוונתם להגיש. לא יתקבלו הגשות נוספות לאחר מועד ההגשה שנקבע בטענות כמו "משום מה הקובץ במודל לא עדכני ויש לנו גרסה עדכנית יותר שלא נקלטה".

בהצלחה!