המעבד בתצורת צנרת

רכיבי המעבד:

I. בין כל 2 שלבים, נכנס אוגר ביניים שמתווך בינהם. אם כן ארבעת אוגרי הביניים IF/ID , ID/EX , EX/MEM , MEM/WB
II. נוספה יחידת העברה קדימה Forwarding Unit
III. נוספה יחידת בקרת סיכונים HDU (Hazard Detection Unit).
IV. נוסף מרבב בין הבקרה הראשית לבין אוגר הביניים ID/EX
V. נוסף משווה ביציאה ממקבץ האוגרים

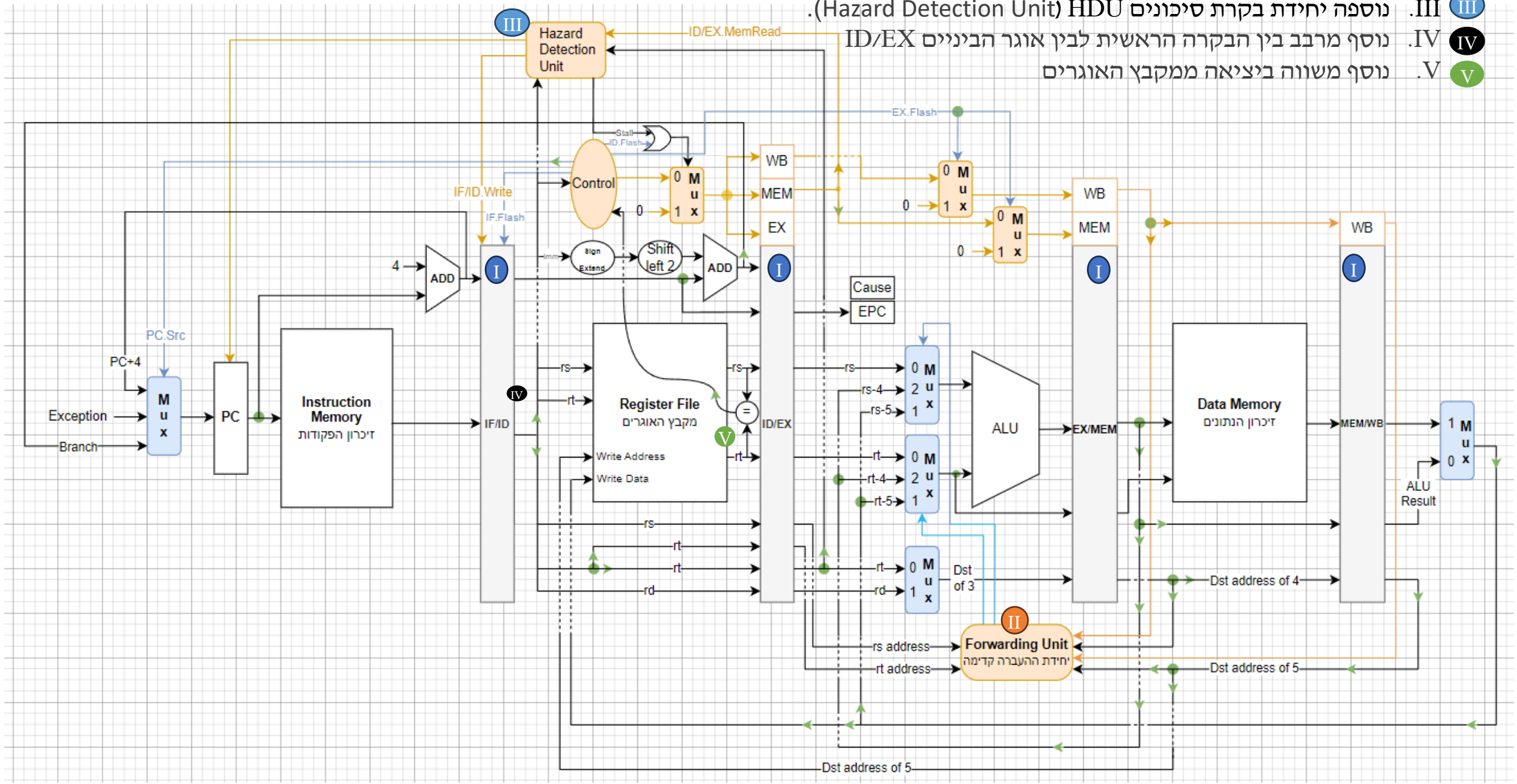
I. בין כל 2 שלבים, נכנס אוגר ביניים שמתווך בינהם. אם כן ארבעת אוגרי הביניים IF/ID , ID/EX , EX/MEM , MEM/WB

II. נוספה יחידת העברה קדימה Forwarding Unit

III. נוספה יחידת בקרת סיכונים HDU (Hazard Detection Unit).

IV. נוסף מרבב בין הבקרה הראשית לבין אוגר הביניים ID/EX

V. נוסף משוות ביציאה ממקבץ האוגרים



מהי תצורת צנרת ומדוע יש בה צורך.

במעבד החד-מחזורי - היו לנו 5 שלבים:

- שלב ההבאה Fetch: שבו מביאים את הפקודה בזיכרון, שהכתובת שלה הייתה באוגר PC.
- שלב הפיענוח Decode: במהלכו מתבצעת קריאה ממקבץ האוגרים ופיענחנו איזו פקודה תבוצע.
- שלב הביצוע Execute: במהלכו מתבצע חישוב ברכיב ה ALU.
- שלב הזיכרון Memory: במהלכו מתבצעת הכתיבה או הקריאה מהזיכרון (אם בכלל)
- שלב ה WB: שבו מתבצעת כתיבה למקבץ האוגרים (אם יש צורך)

לכל שלב כזה, יש משך זמן שלוקח לבצע אותו.

מסלולה של הפקודה במעבד המוצנר:

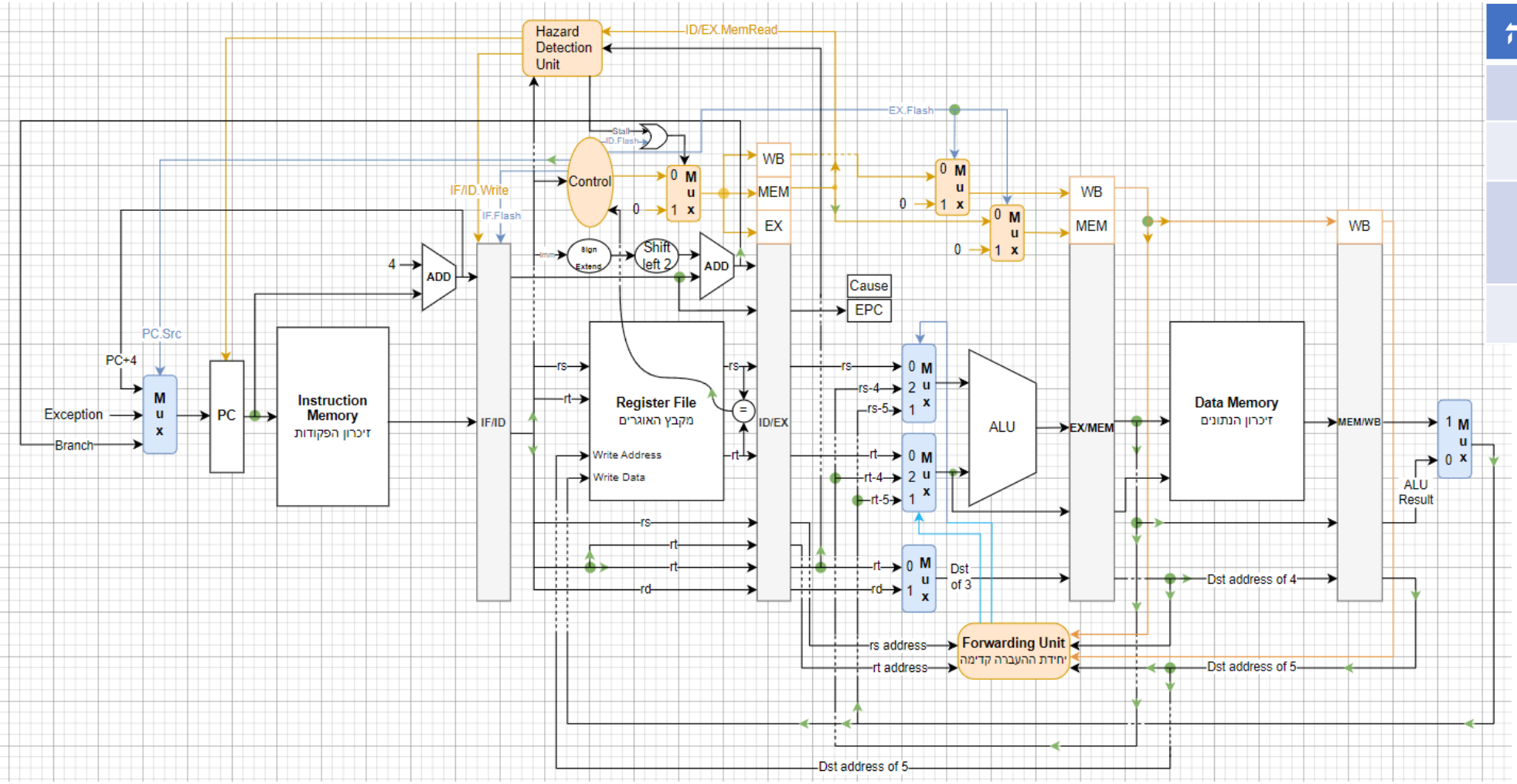
<u>מחזור השעון הראשון</u>: שלב ה Fetch של הפקודה ה lw.

אנו מתחילים כרגיל כאשר האוגר PC מכניס לזיכרון הפקודות את הכתובת של הפקודה הבאה שעלינו לקרוא. הכתובת נכנסת לזיכרון הפקודות שמוציא את קידוד הפקודה ב-32 סיביות.בין כל 2 שלבים נכנס אוגר ביניים חדש. חשוב לדעת, שאוגרים אלו, שונים מעט מהאוגרים שהכרנו עד כה, כמות הסיביות שכל אחד מהם מכיל, גדולה מ 32 סיביות. יחד עם זאת, מספיק לנו לדעת, שהוא שומרים על הערכים שנכניס אליהם כראוי.

הפקודה שקודדה ב-32 סיביות נכנסת לאוגר הביניים הראשון - IF/ID.

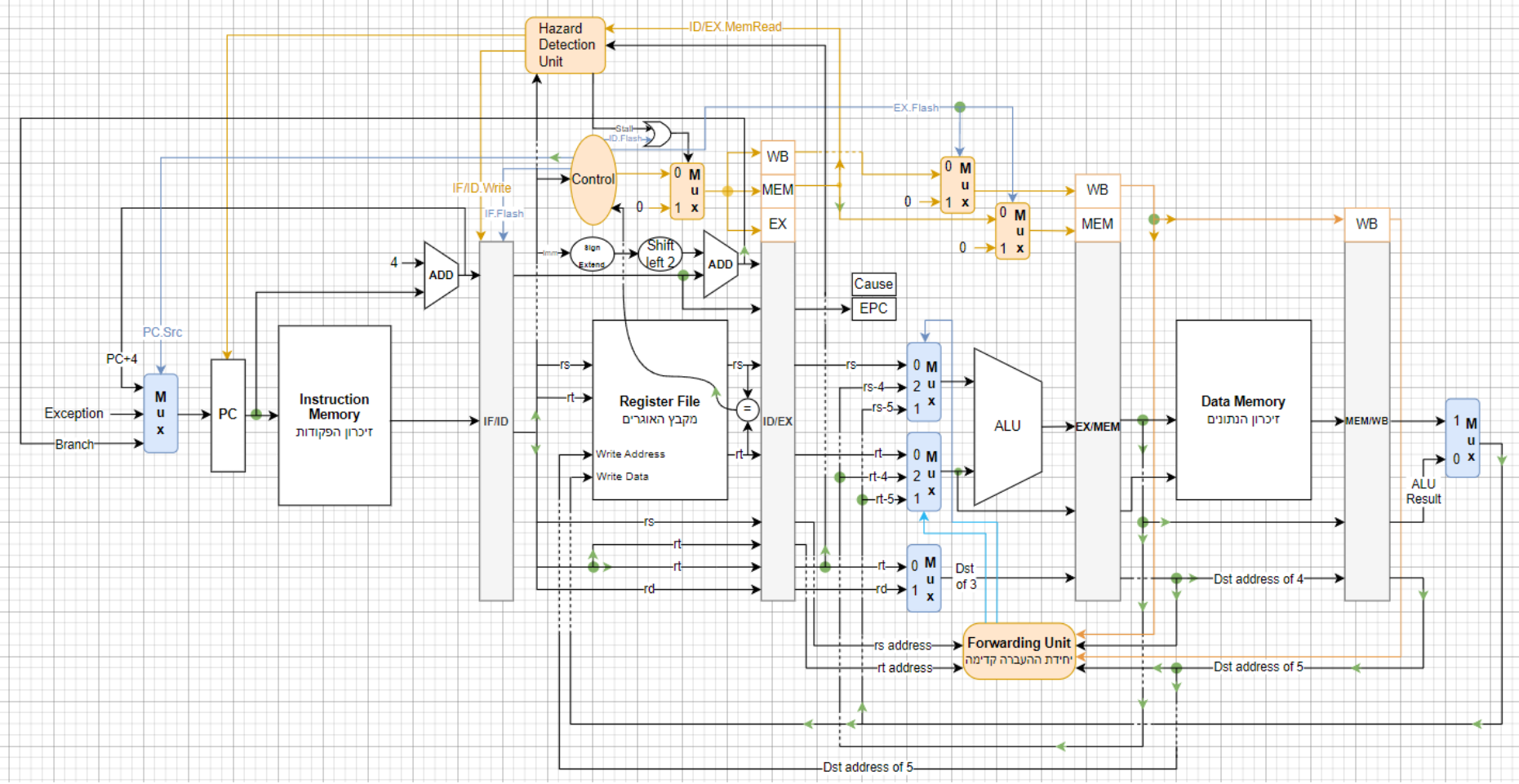
הערה חשובה - מכיוון שבצורה החדשה - בכל שלב מתבצעת פקודה אחרת, אזי עלינו לשמור את קווי הבקרות עבור הפקודות של **כל השלבים**.

לשם כך - כל פקודה מעבירה איתה באמצעות אוגרי הביניים את קווי הבקרות שהיא צריכה. בכל פעם שהפקודה משתמשת בקו בקרה מסוים, היא כבר לא תשמור אותו יותר.



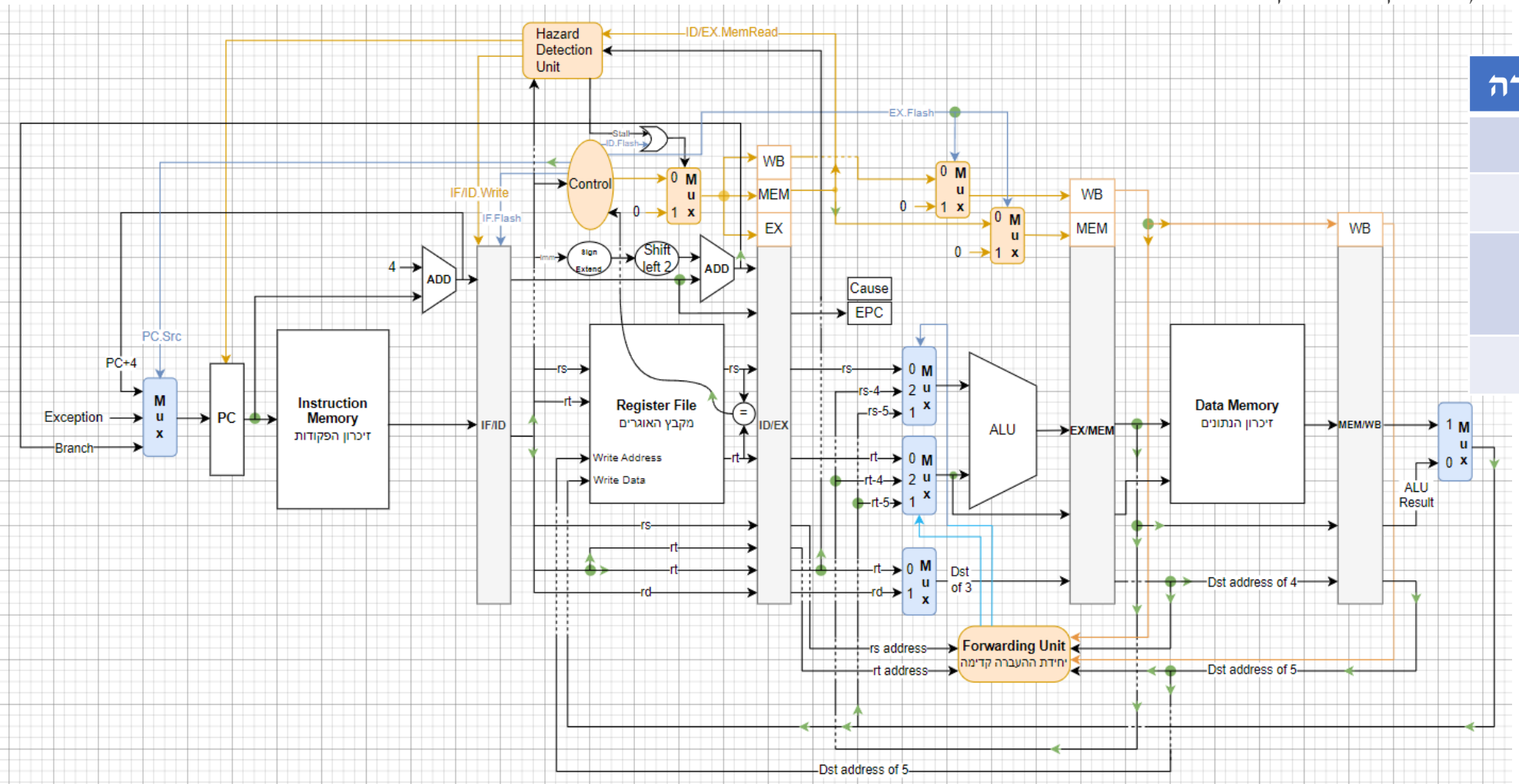
מספר פקודה	פקודה
1	or $11, $12, $13
2	add $8, $9, $10
3	lw $7, 0x64($8)
4	sub $8, $7, $3

מחזור השעון השני: שלב ה Decode של פקודת ה lw וה Fetch של פקודת ה sub.
פקודת ה lw שהתחלנו איתה את המסלול קודם, תיכנס ותבצע את שלב הפענוח - Decode, בעוד שפקודת ה sub
תיכנס לשלב הראשון - Fetch ותתחיל להתבצע גם כן. נמשיך להסביר על פקודת ה lw, ועל מסלולה ובעזרתה נבין איך עובדת המערכת.
בהמשך נתייחס גם לפקודות שקדמו לה, ועל השפעתן עליה.
כעת אנחנו בשלב מספר 2. אוגר הביניים IF/ID יוציא את 32 הסיביות של פקודת ה lw שלנו שנכנסו
אליו וימשיך להזרים אותן הלאה. כרגיל תתבצע קריאה של האוגרים rs,rt ממקבץ האוגרים, אולם מכיוון שבפקודה הנוכחית,
במחזור השעון הנוכחי, לא תתבצע כתיבה למקבץ האוגרים (שהרי אנחנו רק בשלב מספר 2), אזי אין צורך
במרבב שהיה מצוי לפני הכניסה Write Address, של מקבץ האוגרים (כזכור כניסה זו נועדה כדי לקבוע לאיזה אוגר הולכת להתבצע הכתיבה).
במקום זאת, השדות rt ו-rd ימשיכו לזרום גם לשלב הבא. בסיום מחזור שעון זה - כל הערכים שלנו יכנסו לאוגר הביניים שמצוי בין
שלב 2 לשלב 3, כלומר התוכן של האוגר rs, התוכן של האוגר rt, וכן מספרי האוגרים של rt,rd ונוספים יכנסו לאוגר ID/EX.



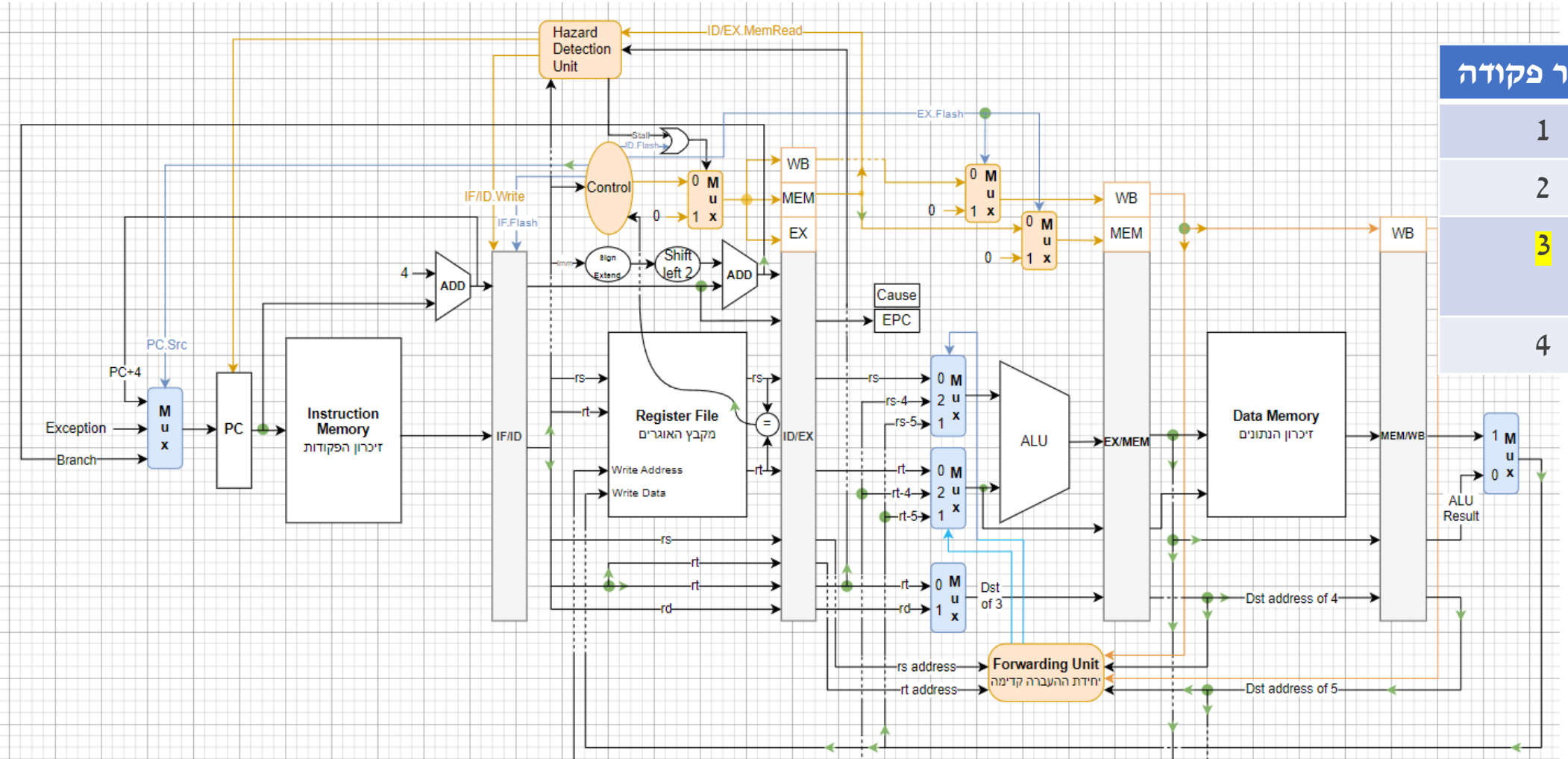
מספר פקודה	פקודה
1	or $11, $12, $13
2	add $8, $9, $10
3	lw $7, 0x64($8)
4	sub $8, $7, $3

מחזור השעון השלישי: לצורך הפשטה, נתמקד תחילה רק ב-3 הפקודות הראשונות.
פקודת or: שלב 5. פקודת add: שלב 4
פקודת lw: שלב 3
כזכור, מספרי האוגרים rt ו rd הוכנסו לאוגר ID/EX? בשלב זה הם יכנסו למרבב שהועבר
משלב 2 - ונועד לקבוע - באיזה אוגר צריכה להתבצע כתיבה (אם בכלל). היציאה שתצא מהמרבב –
תועבר הלאה לאוגר EX/MEM ותשמש אותנו בשלבים הבאים.
כזכור אנו מתמקדמים ב lw שנמצאת בשלב 3. בשלב זה עלינו לבצע חישוב ב ALU שנועד לחבר
את המידע שנמצא בתוך אוגר מספר 8, (ומייצג כתובת) עם המספר 100. סכומם מייצג לנו
כתובת בזיכרון הנתונים שאליה נרצה לגשת, ולאחסן את התוכן של כתובת זו - באוגר מספר 7.



מספר פקודה	פקודה
1	or $11, $12, $13
2	add $8, $9, $10
3	lw $7, 0x64($8)
4	sub $8, $7, $3

אולם כעת אנחנו נתקלים לראשונה בסוגיה, פקודת ה add שקדמה לפקודת ה lw - נמצאת
כעת בשלב מספר 4 והיא אמורה לבצע כתיבה לאוגר מספר 8, אולם היא עדיין לא הספיקה לעדכן אותו.
בו בזמן - פקודת ה lw נמצאת כעת בשלב מספר 3, וצריכה להשתמש באוגר מספר 8 לטובת חישוב הכתובת.
כלומר התוכן שהפקודה lw לקחה מאוגר מספר 8 הוא תוכן לא עדכני. כיצד נוכל להתגבר על בעיה זו?
נבחין בעובדה שהערך החדש של אוגר מספר 8, כבר חושב, ונמצא כרגע באוגר הביניים EX/MEM מכיוון
שבמחזור השעון הקודם, כאשר הפקודה add הייתה בשלב 3 - היא כבר חישבה אותו והכניסה אותו לאוגר זה.
אם כן אם נוכל לקחת את הערך החדש מהאוגר EX/MEM ולהעביר אותו לשלב 3 - הרי שהצלחנו לפתור את הבעיה.
לשם כך נוסף לנו הרכיב הכתום - רכיב ההעברה קדימה שיודע לקחת ערכים עדכניים
משלבים 4 ו 5 ולשים אותם במידת הצורך בכניסות של ה ALU, כיצד הוא עושה זאת?



מספר פקודה	פקודה
1	or $11, $12, $13
2	add $8, $9, $10
3	lw $7, 0x64($8)
4	sub $8, $7, $3

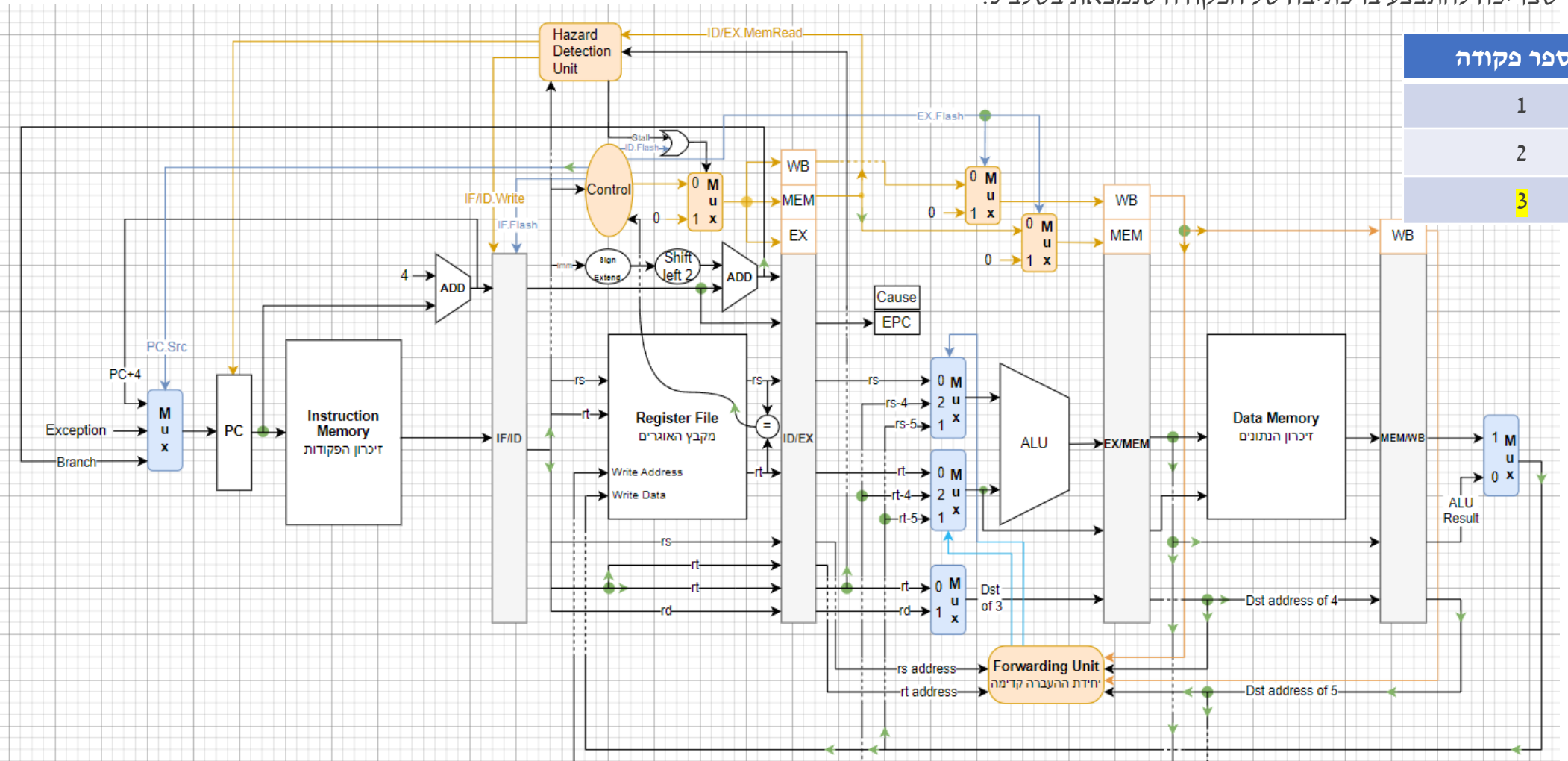
נתבונן בשרטוט - נבחין שלפני הכניסה העליונה ב ALU נוסף מרבב עם 3 כניסות, 3 כניסות אלו מייצגות 3 אפשרויות עבור האוגר rs:

- אפשרות ראשונה: הערך שהגיע ממקבץ האוגרים - אפשרות זו מעידה שהערך במקבץ האוגרים הוא העדכני ביותר.
- אפשרות שנייה: הערך העדכני ביותר - נמצא בשלב מספר 4 (כמו בדוגמה שעליה דיברנו עם פקודת ה add)
- אפשרות שלישית: הערך העדכני ביותר - נמצא בשלב מספר 5 (למשל ערך שהגיעה מהזיכרון).

נתבונן ביחידת ההעברה קדימה:

- היא מקבלת את מספר האוגר של rs ואת מספר האוגר של rt של הפקודה בשלב מספר 3.
- היא מקבלת את קו הבקרה RegWrite של הפקודה משלב 4 , ואת RegWrite של הפקודה משלב 5.
- היא מקבלת את מספר האוגר שצריכה להתבצע בו כתיבה של הפקודה שנמצאת בשלב 4.
- ומקבלת את מספר האוגר שצריכה להתבצע בו כתיבה של הפקודה שנמצאת בשלב 5.

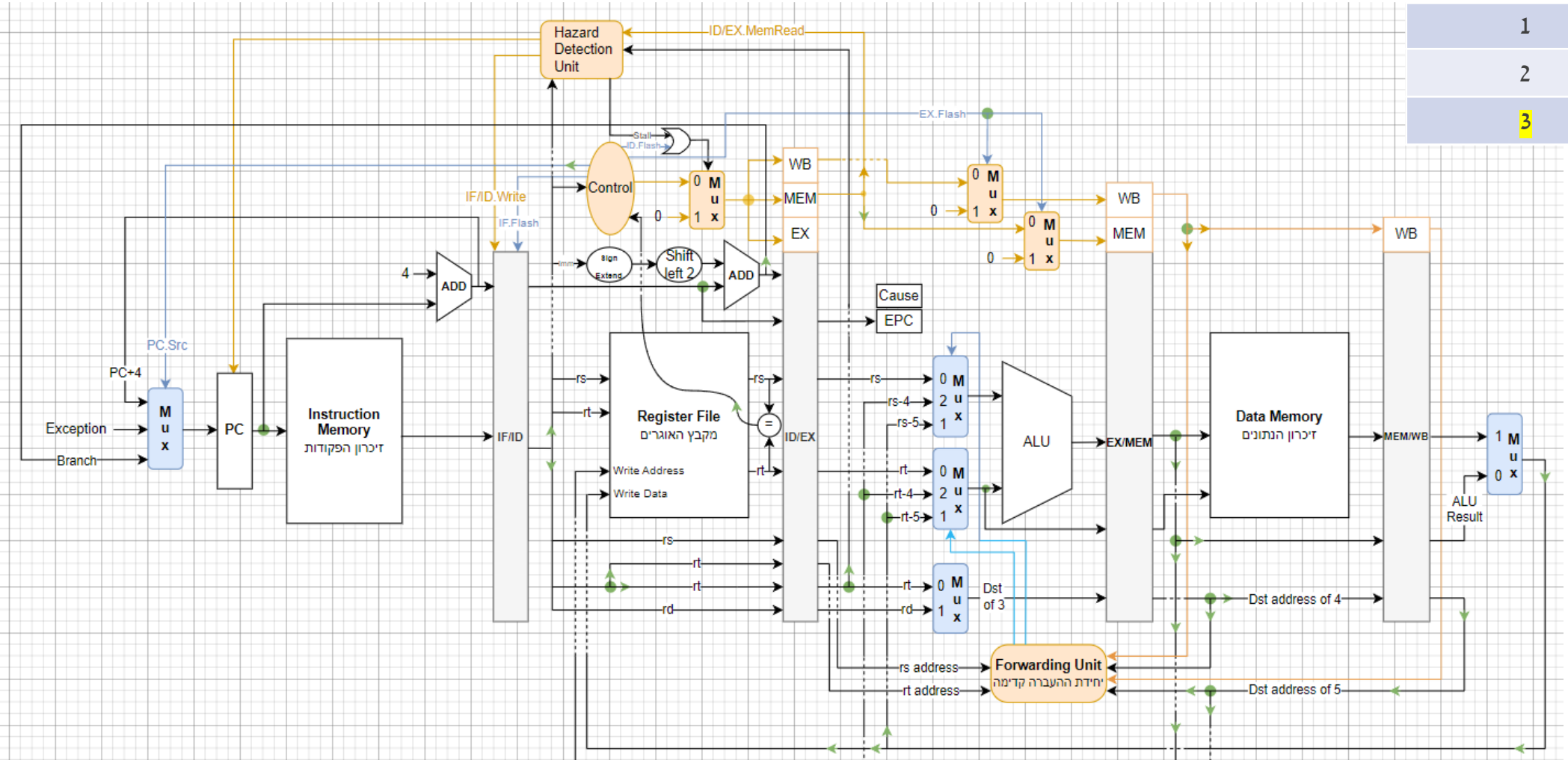
פקודה	מספר פקודה
or $11, $12, $13	1
add $8, $9, $10	2
lw $7, 100($8)	3



כיצד יחידת ההעברה קדימה יודעת לבחור?

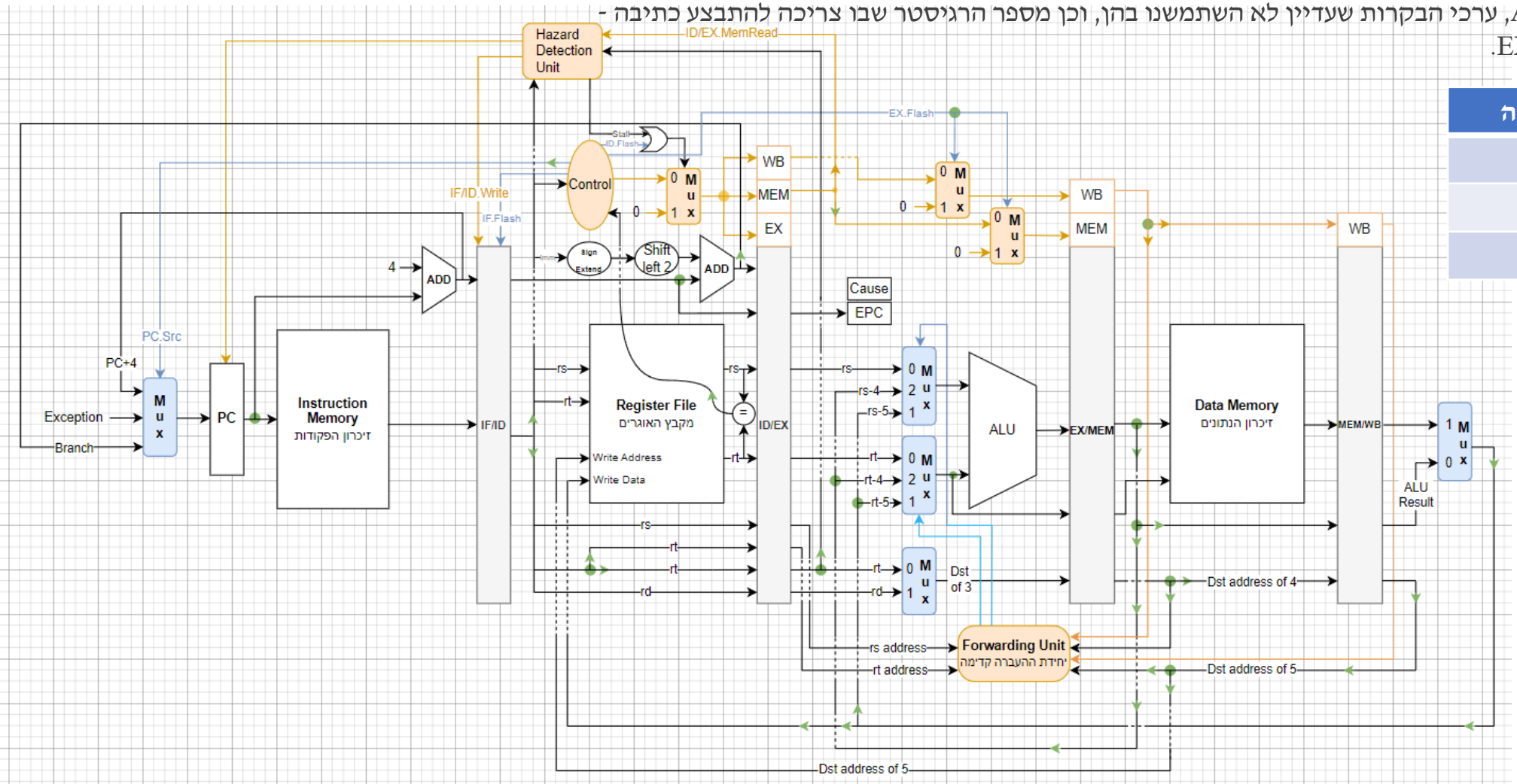
- היא בודקת האם הפקודה משלב 4 אמורה בכלל לבצע כתיבה במקבץ האוגרים. (אחרת בכלל אין התנגשות)
- היא בודקת האם מספר האוגר rs של שלב 3 שצריך להיכנס ל ALU הוא אותו מספר אוגר שהפקודה משלב 4 צריכה לכתוב בו.
- אם התשובה ל-2 השאלות הללו היא "כן" אז יש בשלב 4 ערך עדכני יותר, ולכן יחידת ההעברה קדימה תיכנס לפעולה, ותבחר באפשרות האמצעית של המרבב - את שמה של כניסה זו סימנו להיות "rs-4" כדי לסייע לכם התלמידים לזכור, שמדובר בערך עדכני של האוגר rs, שהגיע משלב מספר 4.
- בדיקה זהה - תתבצע ותבדוק האם יש צורך לבצע העברה קדימה משלב 5. לשם כך יש למרבב גם את הכניסה "rs-5".

פקודה	מספר פקודה
or $11, $12, $13	1
add $8, $9, $10	2
lw $7, 100($8)	3



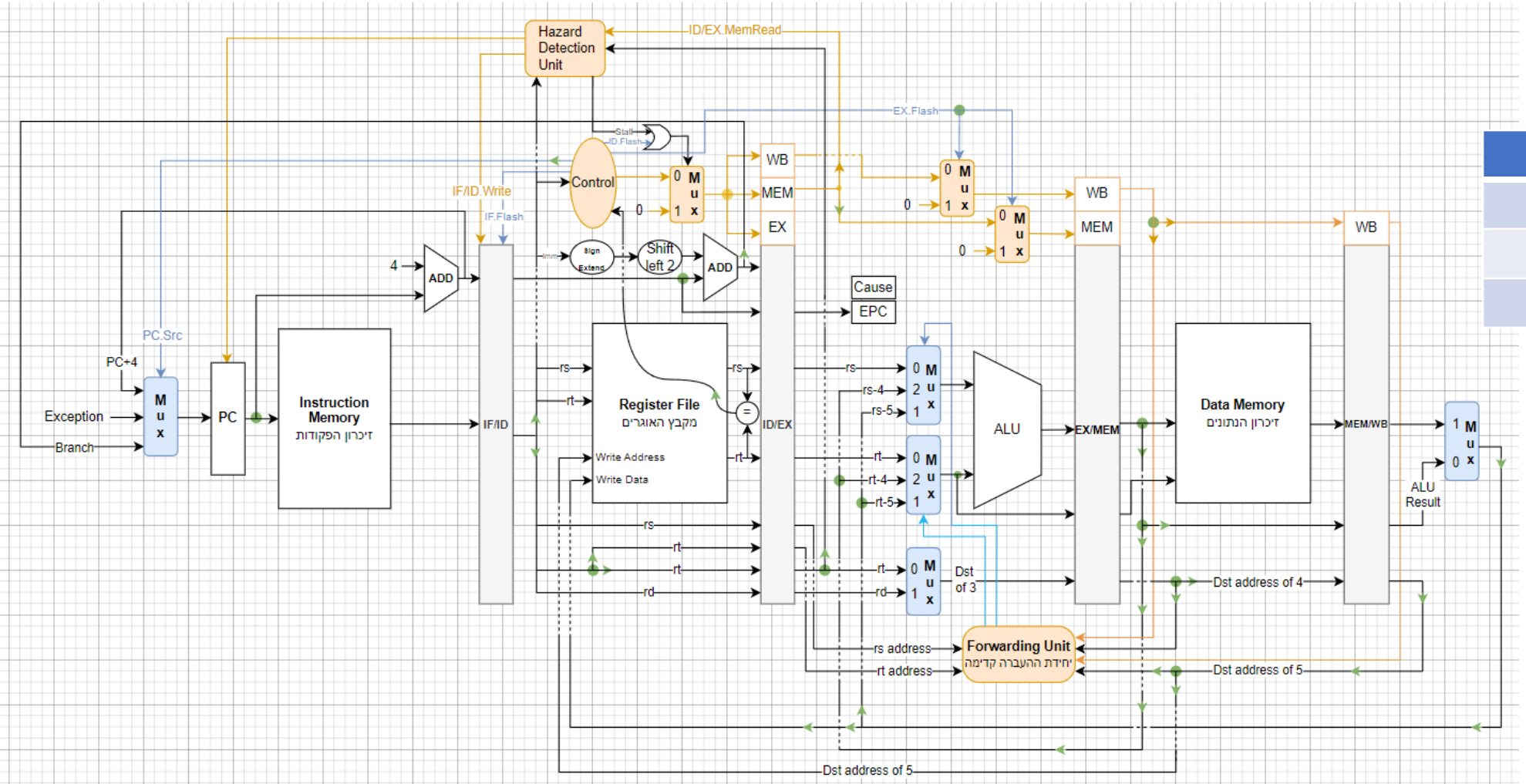
- לסקרנים שבינכם השואלים - מה יקרה אם יש יהיה ערך עדכני לאוגר מספר 8 גם בשלב מספר 4 וגם בשלב מספר 5, אל דאגה! נזכור שבמצב זה הערך משלב 4 הוא העדכני יותר מבין שניהם - וזאת מכיוון שהוא מגיע מאוחר יותר כרונולוגית. ולכן הערך שמגיע ממנו ייבחר.
- ❖ בצורה דומה כמו שבדקנו האם יש ערך עדכני יותר עבור הכניסה של rs ל ALU, תתבצע גם בדיקה מקבילה על האם יש ערך עדכני יותר עבור הכניסה של rt ל ALU, ובמידת הצורך - תתבצע העברה קדימה גם שם.
- ❖ בדוגמה שהצגנו - תתבצע העברה קדימה של ערך אוגר 8 העדכני מפקודת ה add בשלב 4 לפקודת ה lw שבשלב 3.
- ❖ כמו כן נשים לב שקווי הבקרות של שלב ה EX בהן נעשה שימוש - לא ימשיכו לזרום הלאה.
לדוגמה קו הבקרה של המרבב שאינו מופיע בשרטוט, ובוחר בין ערכי rt האפשריים לבין הערך המיידי שומש ולכן אין בו עוד צורך.
- ❖ בסיום שלב זה - תוצאת ה ALU, ערכי הבקרות שעדיין לא השתמשנו בהן, וכן מספר הרגיסטר שבו צריכה להתבצע כתיבה - יועברו לאוגר הביניים EX/MEM.

פקודה	מספר פקודה
or $11, $12, $13	1
add $8, $9, $10	2
lw $7, 100($8)	3



מחזור השעון הרביעי:

בשלב 4 בו אנו נמצאים כעת, אין שינוי מיוחד בהשוואה למעבד החד מחזורי, תתבצע קריאה מזיכרון הנתונים.
כמו כן נבחין שהערך שיצא מאוגר הביניים EX/MEM והגיע במקור מה ALU כשהפקודה הייתה בשלב 3 –
חוזר גם אחורה למרבבים שנמצאים בכניסות של ה ALU לטובת אפשור של העברה קדימה במידת הצורך כמו שהסברנו בשלב הקודם.
בסיום מחזור שעון זה - המידע שהגיע מזיכרון הנתונים, תוצאת ה ALU וכן מספר האוגר
שבו צריכה להתבצע כתיבה מועברים כולם לאוגר הביניים MEM/WB. נשים לב שוב שמכיוון שהשתמשנו
בערכי הבקרות שקשורים לזיכרון - MemWrite ו MemRead, כבר אין לנו צורך בהם, ולכן הם אינם
מועברים לקראת שלב 5, אלא רק הבקרות של שלב ה WB.

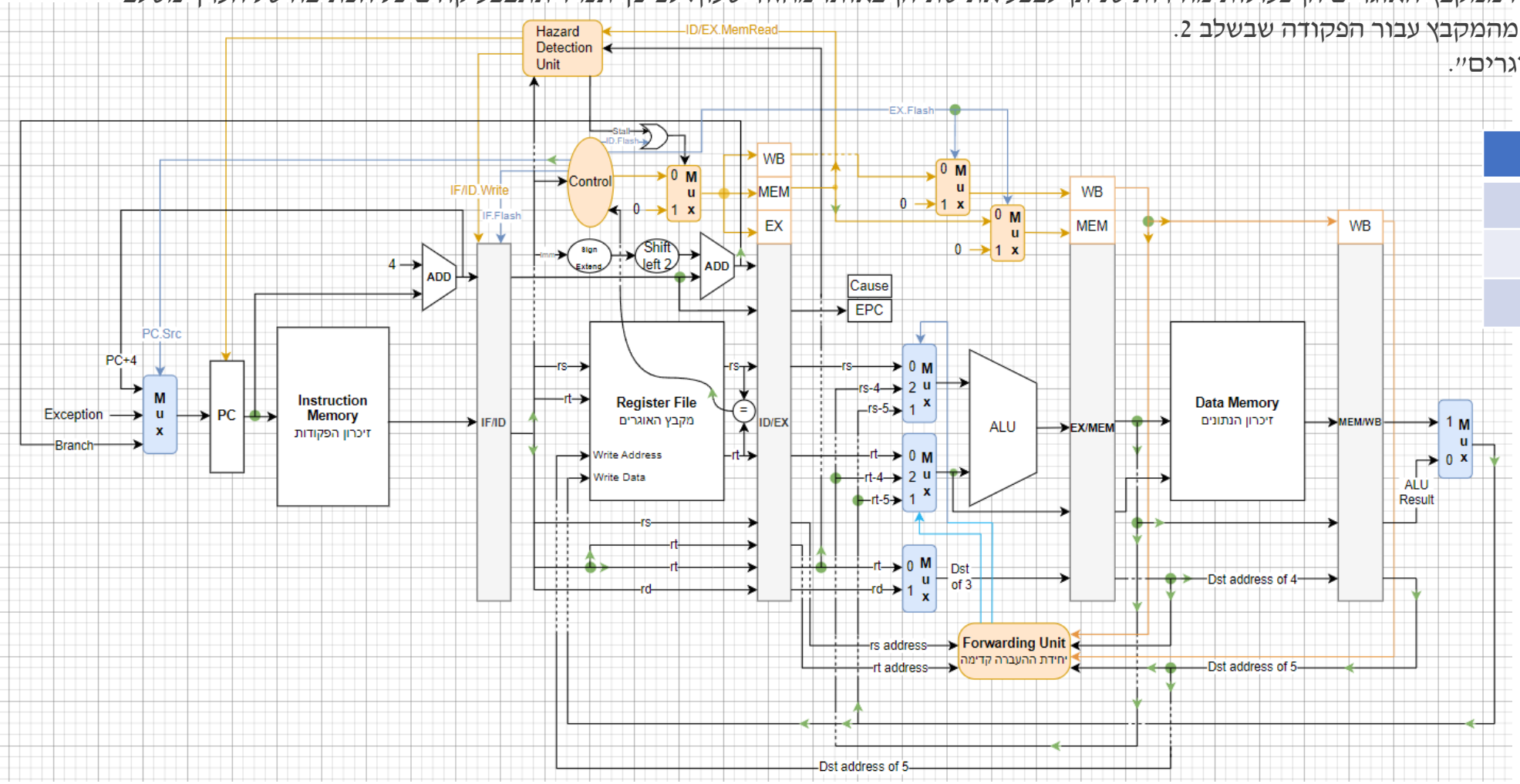


מספר פקודה	פקודה
1	or $11, $12, $13
2	add $8, $9, $10
3	lw $7, 100($8)

מחזור השעון החמישי:

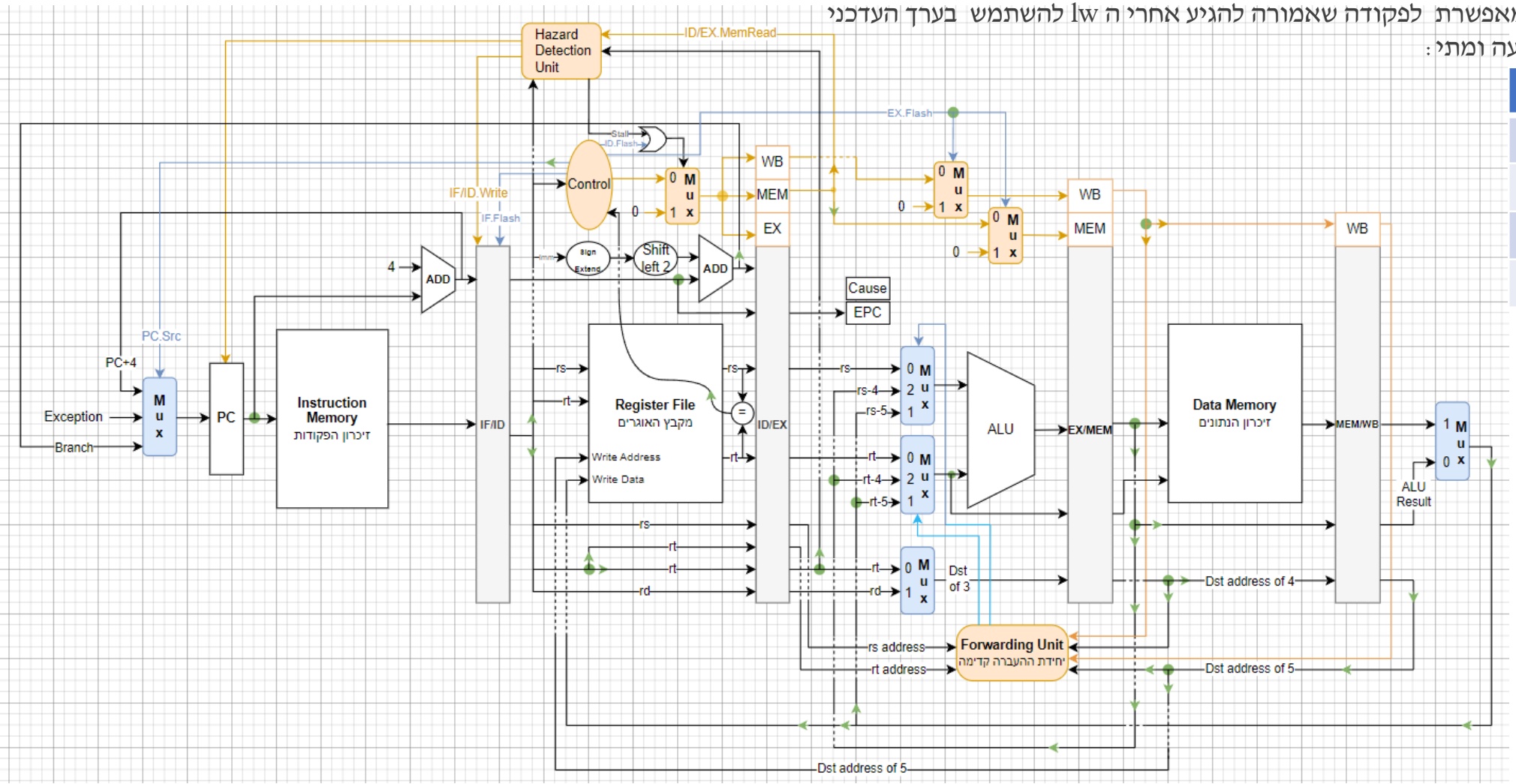
בשלב 5 בו אנו נמצאים כעת, תתבצע כרגיל בחירה מהיכן לקחת את המידע שייכתב במקבץ האוגרים –
האם מזיכרון הנתונים או מהערך שיצא מה ALU, הבחירה תיעשה באמצעות קו הבקרות שלקחנו איתנו מאוגר הביניים.
בדוגמה שלנו תיבחר כמובן האפשרות של מידע מזיכרון הנתונים (שהרי מדובר בפקודת lw).
כמו כן מספר האוגר שאליו צריכה להתבצע כתיבה, אותו לקחנו איתנו - יכנס בחזרה למקבץ האוגרים, ובו הערך ייכתב.
קו הבקרות הנוסף שלקחנו איתנו הוא RegWrite והוא יכנס למקבץ האוגרים כדי לסמן לו האם בכלל צריכה להתבצע כתיבה.

- עולה השאלה איך יכול להיות שבאותו שעון גם תתבצע כתיבה במקבץ האוגרים (בגלל הפקודה שנמצאת בשלב מספר 5) ו
גם תתבצע קריאה ממקבץ האוגרים של הפקודה שנמצאת בשלב מספר 2? ובפרט מה יקרה אם מדובר על אותו אוגר שצריך גם לכתוב בו וגם לקרוא ממנו?
- התשובה לכך היא שכתיבה / קריאה ממקבץ האוגרים הן פעולות מהירות שניתן לבצע את שתיהן באותו מחזור שעון. לפיכך תמיד תתבצע קודם כל הכתיבה של הערך משלב
5, ורק לאחר מכן קריאה של הערך מהמקבץ עבור הפקודה שבשלב 2.
פיתרון זה נקרא "חציית מקבץ האוגרים".



מספר פקודה	פקודה
1	or $11, $12, $13
2	add $8, $9, $10
3	lw $7, 100($8)

נחשוב לרגע על פיתרון ההעברה קדימה שיצרנו. הוא נועד לאפשר לנו להמשיך להריץ את הפקודות כך שהן יקבלו
את הערכים העדכניים שהן צריכות, גם מבלי לחכות עד לסיומן. לעיתים העברה קדימה משלב 4/5 לשלב מספר 3 אינה מספיקה,
מכיוון ששורת הקוד הבאה, צריכה את המידע של הערך העדכני בשלב **מוקדם** אף יותר משלב 3. למשל בדוגמה שלנו –
הערך החדש שאמור להיכתב באוגר מספר 7 בפקודת ה lw, יצא מזיכרון הנתונים רק ב**סיום** מחזור השעון הרביעי,
אך פקודת ה sub שבאה אחריה ונמצאת לכאורה בשלב השלישי צריכה את הערך החדש של אוגר מספר 7
כבר בתחילת מחזור השעון הרביעי, ולכן העברה קדימה לא תעזור במקרה זה. סיכון נתונים ספציפי זה שנקרא load use –
יכול לקרות רק בפקודות lw ועל כן כדי למנוע אותו, עלינו להכניס **השהייה**, ובאנגלית stall. ההשהייה מכונה
גם בועה bubble והיא בעצם הכנסה של פקודת סרק שלא עושה כלום,למשך מחזור שעון אחד לאחר פקודת ה lw,
ובזכות ההשהייה שהיא מביאה - היא מאפשרת לפקודה שאמורה להגיע אחרי ה lw להשתמש בערך העדכני
ולא בערך שגוי. נסביר כיצד עובדת הבועה ומתי:



פקודה בתכנון	מספר פקודה
or $11, $12, $13	1
add $8, $9, $10	2
lw $7, 100($8)	3
sub $8, $7, $3	4

פקודה בפועל	מספר פקודה
or $11, $12, $13	1
add $8, $9, $10	2
lw $7, 100($8)	3
השהייה	4
sub $8, $7, $3	5

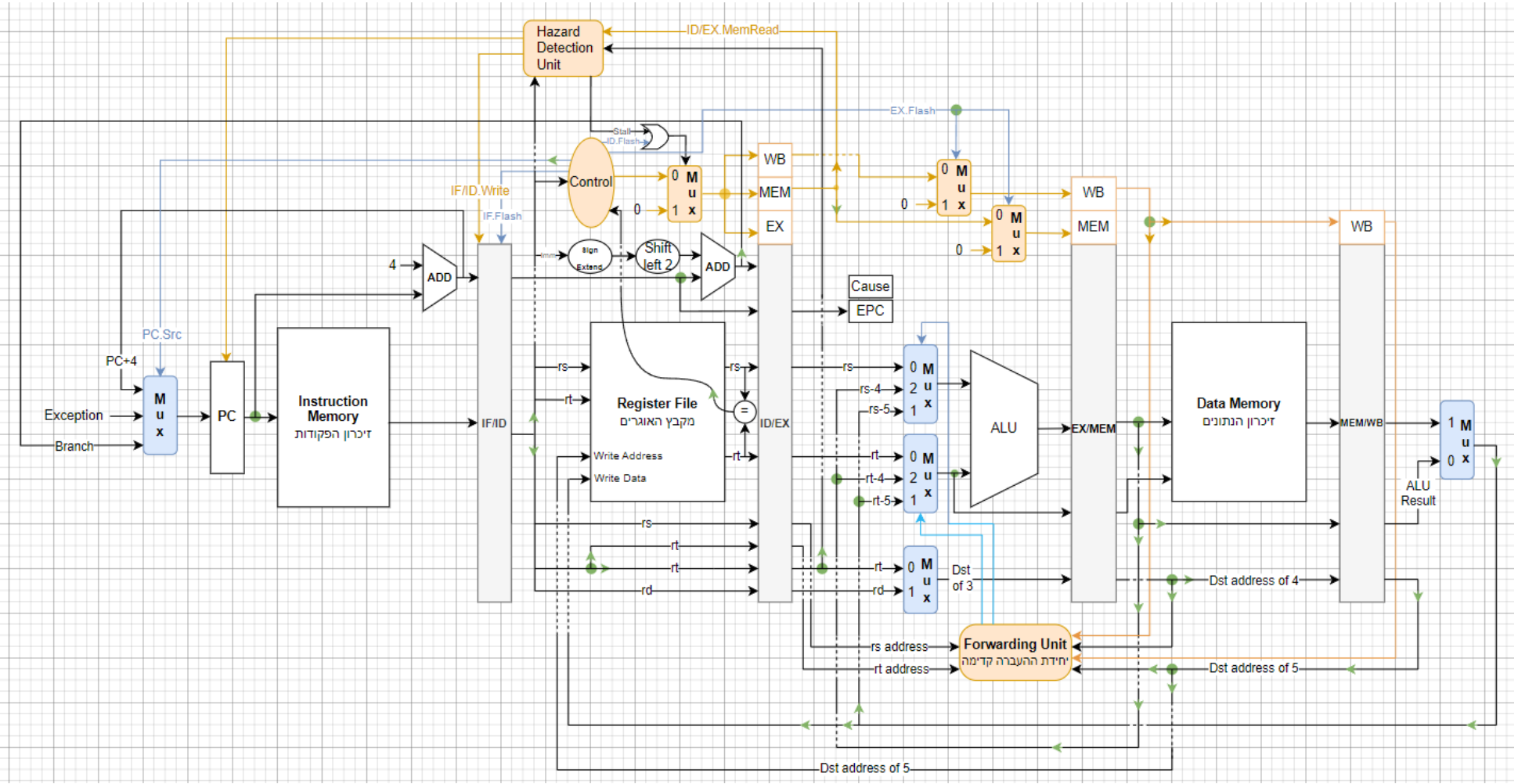
לשם כך נדבר כעת על רכיב חדש - יחידת איתור הסיכונים - HDU. : יחידה זו מקבלת את קו הבקרה MemRead של הפקודה שנמצאת בשלב 3, את מספר האוגר rt של הפקודה בשלב מספר 3, ואת מספרי האוגרים rs,rt של הפקודה משלב 2. בועה תיכנס בהינתן התנאים הבאים :

- הפקודה שנמצאת בשלב 3 צריכה לקרוא מהזיכרון (פקודה כמו lw) לכן הבקרה MemRead = 1
- מספר האוגר שהפקודה משלב 3 (ה lw) צריכה לכתוב בו הוא אחד מהאוגרים rs/rt שהפקודה שנמצא בשלב 2 צריכה להשתמש בהם.

ומה בעצם אומרת בועה? בועה היא איפוס של כל קווי הבקרות שהפקודה משלב 2 תיקח איתה לכל אורך הדרך – כלומר בפרט לא תתבצע כתיבה בזיכרון או כתיבה במקבץ האוגרים ולכן הפקודה לא תגרום לשום שינוי מעשי.

אולם חשוב לדעת שהערכים של הפקודה בשלב 2 שהופכת לבועה כן ממשיכים לזרום הלאה במעבד וכן עוברים חישובים ב ALU.

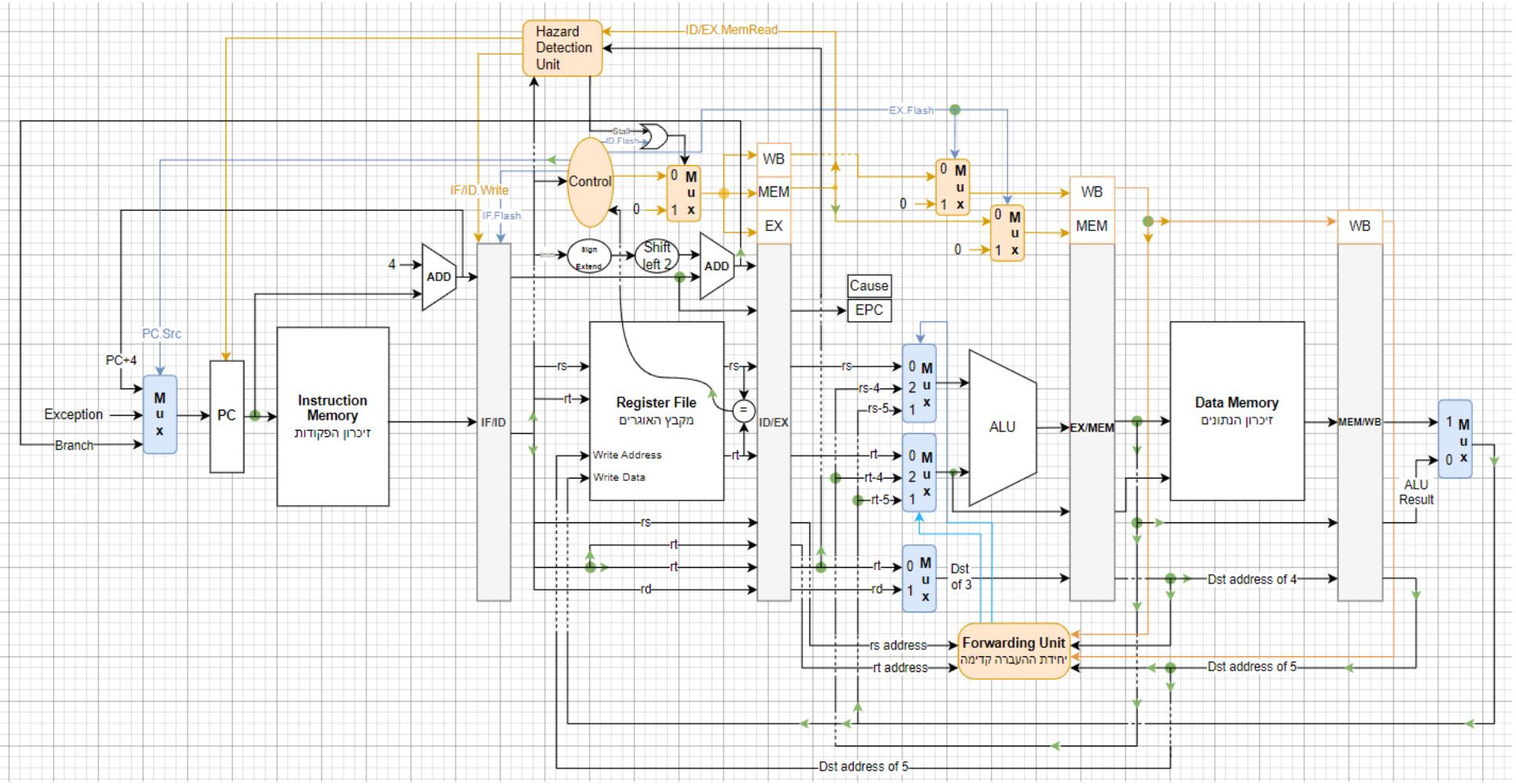
איפוס קווי ייעשה ע"י מרבב שנמצא בשלב 2 ומחליט איזה ערכי בקרות להעביר הלאה האם לקחת את קווי הבקרות הרגילים של הפקודה משלב 2? או לחילופין לשים '0' בכל הבקרות לטובת השהייה.



מספר פקודה	פקודה בתכנון
1	or $11, $12, $13
2	add $8, $9, $10
3	lw $7, 100($8)
4	sub $8, $7, $3

מספר פקודה	פקודה בפועל
1	or $11, $12, $13
2	add $8, $9, $10
3	lw $7, 100($8)
4	השהייה
5	sub $8, $7, $3

כעת נדבר על סיכוני בקרות, ועל פתרונם כפי שמוצג בתצורת המעבד הסופית שנלמדת בקורס.
סיכוני בקרות קיימים במקומות שבהם יש פקודת קפיצה לכתובת אחרת. במקרה זה
הפקודה שמגיעה אחרי פקודת הקפיצה לא צריכה להתבצע. נתבונן בדוגמה שמופיעה בצד שמאל,
ונניח שערכי האוגרים $11 ו- $12 הם 5. במקרה זה הקפיצה של פקודת ה beq צריכה להתבצע מכיוון שהאוגרים שווים.
אולם לעומת המעבד החד מחזורי, בדיקת השוויון שמצריכה פקודת ה beq הוקדמה משלב 3 לשלב 2,
וזאת כדי לצמצם את כמות ההשהיות שפקודת זו תצריך. אם כן נקודת הזמן עליה נתבונן כרגע היא כאשר פקודת ה beq נמצאת בשלב מספר 2,
כזכור, כן צריכה להתבצע קפיצה לפקודה מספר 5, אולם בינתיים הספיקה פקודה מספר 2 - ה add להיכנס לשלב מספר 1,
זו כמובן בעיה שהרי פקודה זו לא אמורה לזרום בצנרת, מה נוכל לעשות כדי להימנע מטעויות?
מכיוון שהפקודה כבר נכנסה למערכת, לא נוכל להוציא אותה החוצה, אך נוכל לתת לה להמשיך לזרום בצנרת מבלי שתהיה לה השפעה על הזיכרון ועל האוגרים, נוכל לעשות זאת על ידי איפוס
כל הסיביות שלה, כלומר קידוד הפקודה שתתקבל יהיה 32 אפסים. פעולה זו נקראת "שטיפה - flush".



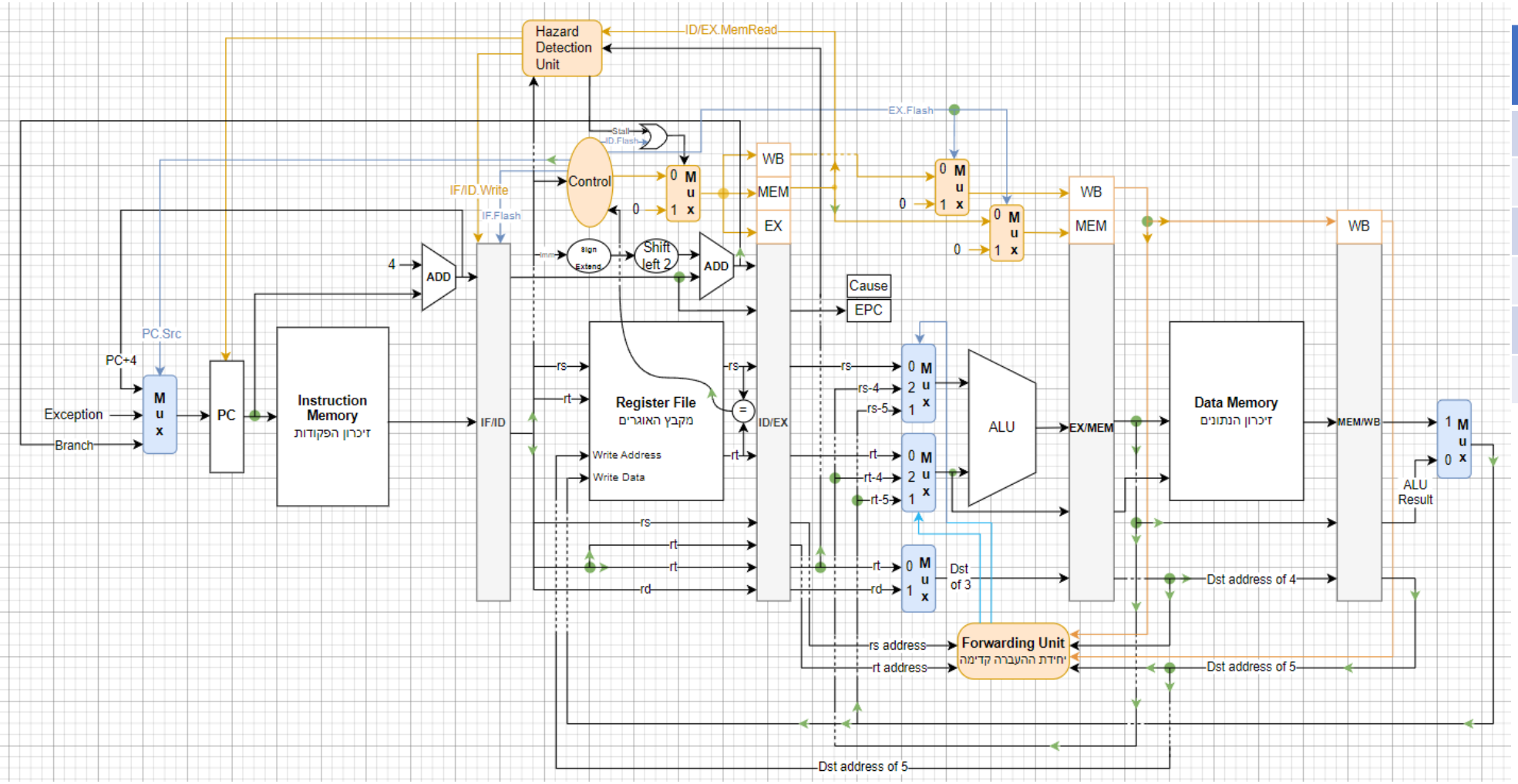
מספר פקודה	פקודה
1	beq $11, $12, L2
2	add $8, $9, $10
3	sub $8, $9, $10
4	sw $5, 20($12)
5	L2: add $8, $9, $10
6	sub $8, $7, $3

הבקרה הראשית היא האחראית על ביצוע השטיפה לפקודה שמגיעה אחרי ה beq במידת הצורך, נסביר איך היא עושה זאת:

- בעזרת ה opcode שהיא מקבל היא תדע האם זו פקודה מסוג beq.
- התוצאה של המשווה שנמצא בשלב 2 ומשווה בין 2 האוגרים, נכנס לבקרה גם כן, כך היא יודעת האם צריכה להתבצע קפיצה.

אם התשובה ל-2 שאלות אלו היא "כן" אזי עלינו לבצע שטיפה, ולהפוך את כל הסיביות של הפקודה שתיכנס במחזור השעון הבא לשלב מספר 2 יהיו 0. זה נעשה כך:

- קו הבקרה IF/Flush יאפס בסוף פעימת השעון את האוגר IF/ID - כתוצאה מכך הסיביות שיוזרמו לשלב 2 יהיו אפסים.
- המרבב PC Src שנמצא לפני אוגר ה PC יבחר הפעם את כתובת הקפיצה שהגיעה מהפקודה beq.



מספר פקודה	פקודה
1	beq $11, $12, L2
2	add $8, $9, $10
3	sub $8, $9, $10
4	sw $5, 20($12)
5	L2: add $8, $9, $10
6	sub $8, $7, $3

חריגות:

חריגות (exceptions) - אירוע לא צפוי במעבד, לדוגמה:

- opcode לא מוגדר - זיהוי של חריגה כזו תקרה בשלב 2
- גלישה בפעולת חיבור או חיסור - זיהוי של חריגה כזו תקרה בשלב 3

פסיקות (interrupts) - אירוע לא צפוי במעבד, שנגרם מסיבה חיצונית.

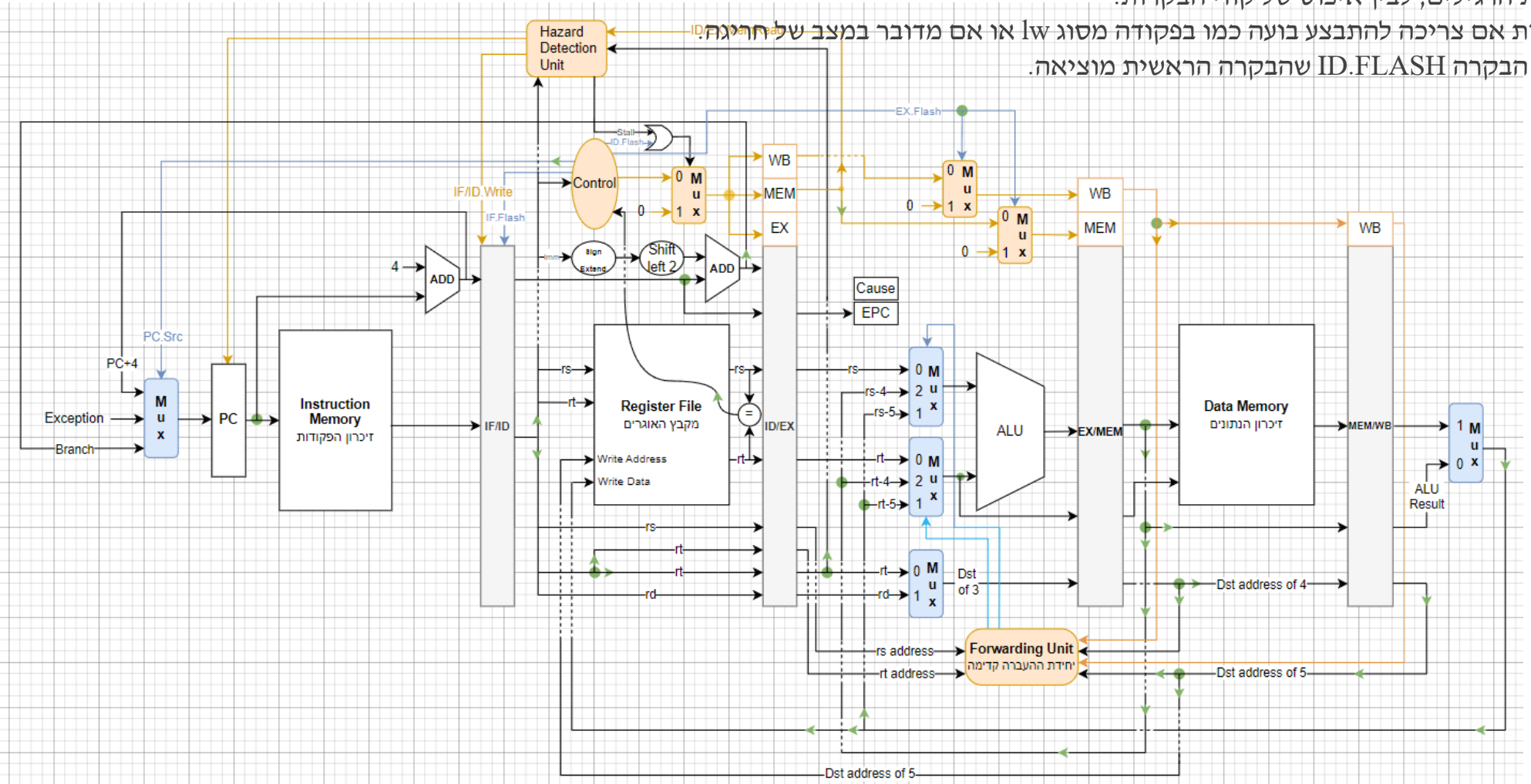
לאחר זיהוי החריגה, עלינו לשטוף גם את הפקודה שבה התבצעה החריגה, וגם את הפקודות שכבר הספיקו להיכנס למערכת אחריה.

- כדי לשטוף את הפקודה שנמצאת בשלב 1, נהפוך אותה לפקודת אפסים (nop) כפי שעשינו בפקודות beq.
- כדי לשטוף פקודה שנמצאת בשלב 2, נוסף שער OR לפני המרבב שקובע את ערכי קווי הבקרות.

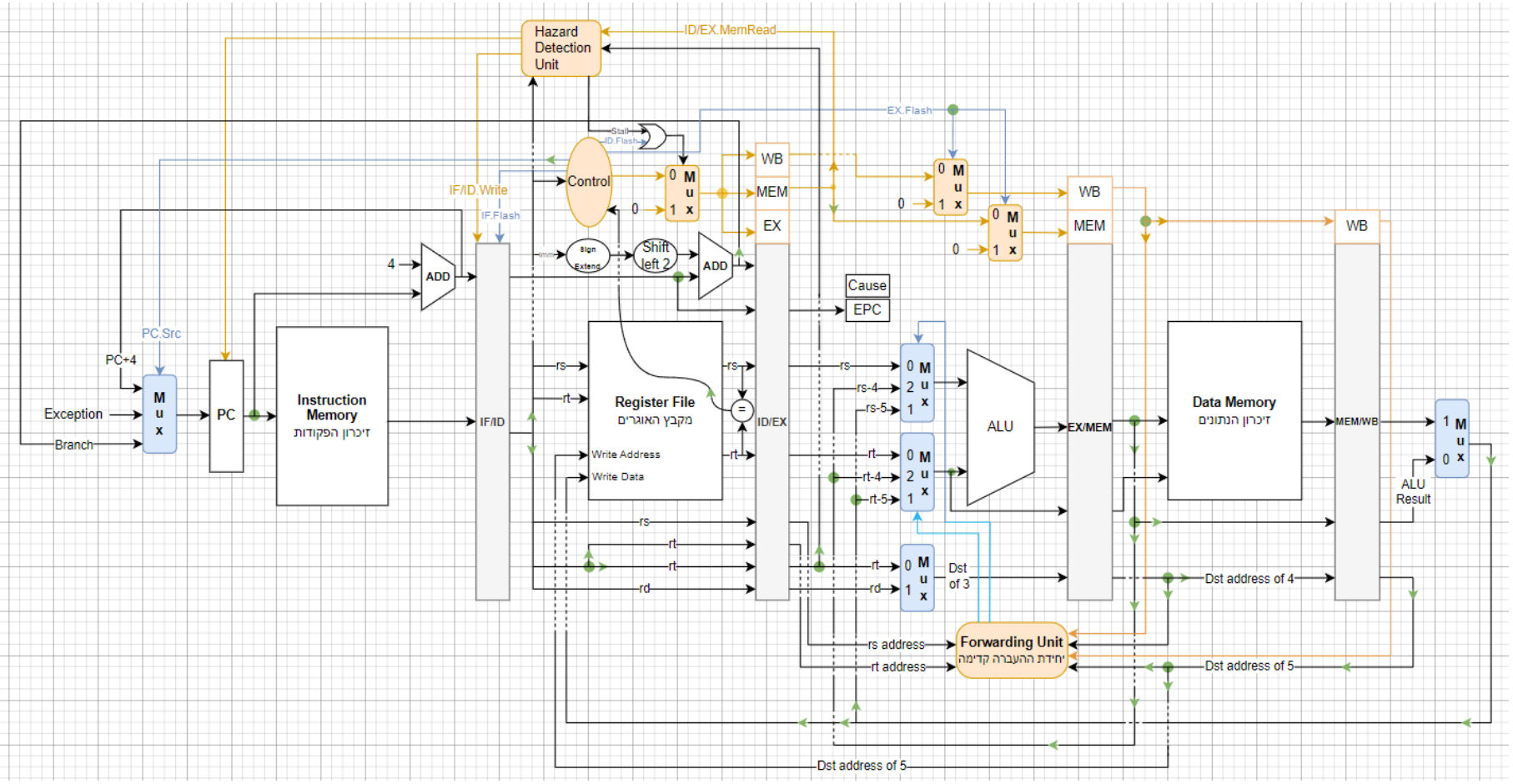
כזכור מרבב זה בוחר בין קווי הבקרות הרגילים, לבין איפוס של קווי הבקרות.

שער ה or יבחר לאפס את קווי הבקרות אם צריכה להתבצע בועה כמו בפקודה מסוג lw או אם מדובר במצב של חריגה.

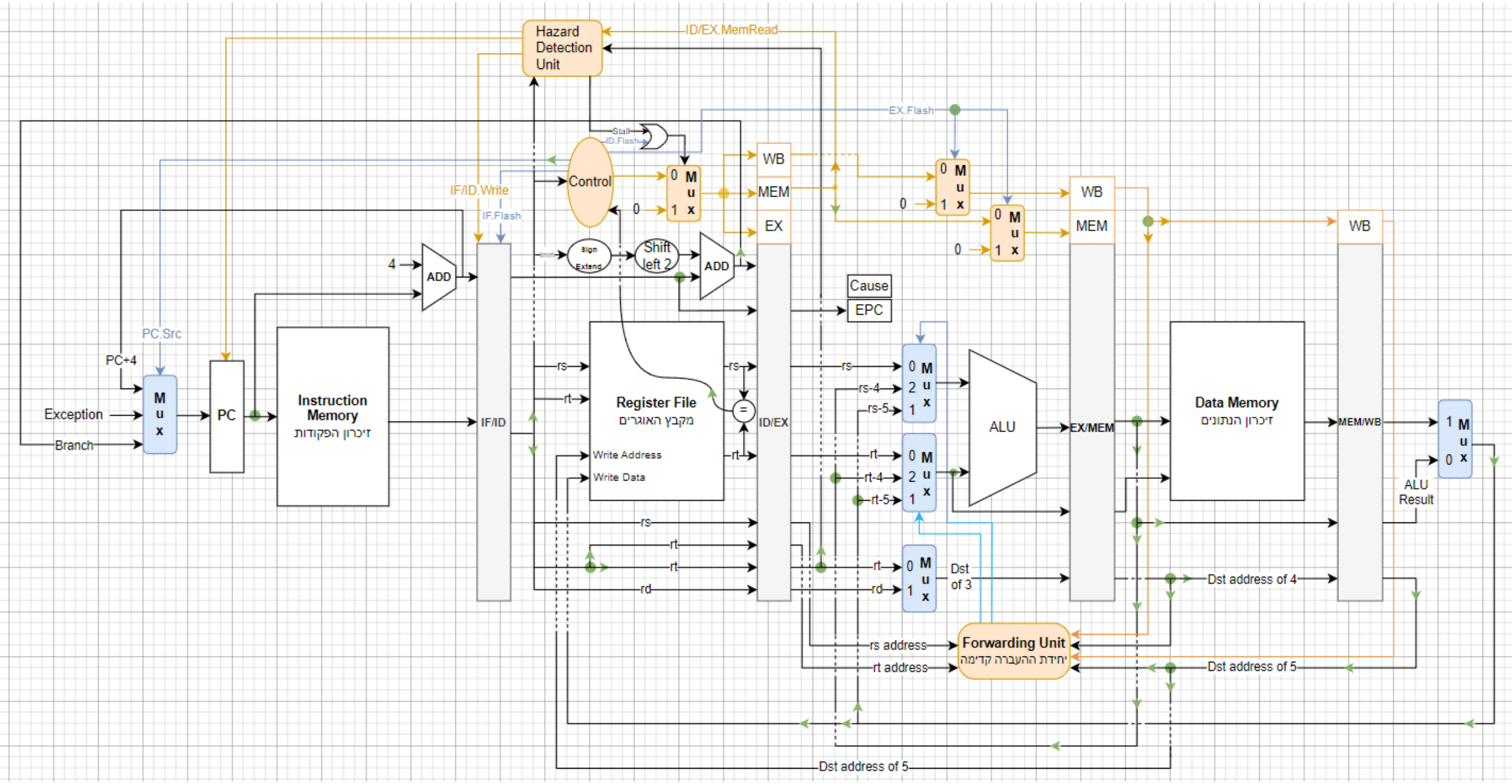
מה שיעיד על מצב של חריגה יהיה קו הבקרה ID.FLASH שהבקרה הראשית מוציאה.



- אם זיהוי החריגה היה בשלב 3 (למשל בגלישה של חיבור) אז נרצה לשטוף גם אותה, לשם כך נוספו 2 מרבבים:
  - מרבב עבור קווי הבקרות שמיועדות לשלב ה Mem ובוחר בין הקווים הרגילים לבין 0.
  - מרבב עבור קווי הבקרות שמיועדות לשלב ה WB ובוחר בין הקווים הרגילים לבין 0.
  - ❖ קו הבקרה שיגרום למרבבים לבחור נכונה - הוא קו חדש שנקרא EX.Flush והוא יפעל כשיש חריגה.
- נשים לב גם שלמרבב שבוחר את כתובת הפקודה הבאה עבור PC נוספה הכתובת 0x80000180 שהיא כתובת בזיכרון שמיועדת לטיפול בחריגות.
- במקרה שיש חריגה, נרצה לשמור את הכתובת של הפקודה שגרמה לחריגה. הכתובת תישמר באוגר שנקרא EPC.
נשים לב שהכתובת של הפקודה שבה אירעה החריגה כבר עברה הגדלה ב 4, ולכן נצטרך לחסר ממנה 4 כדי לחזור לכתובת המקורית.
- יחידת הבקרה תוציא כפלט גם את הסיבה לחריגה, זה יתבצע באוגר Cause. במימוש המצומצם שלנו יש רק 2 סיבות לחריגה, ולכן ניתן להסתפק במרבב נוסף שיכנסו אליו 2 מספרי החריגות האפשריות, קו בקרה למרבב שיבחר איזה מספר חריגה, וקו בקרה שמאפשר כתיבה לאוגר ה Cause. התאמת חומרה זו איזה מופיעה בשרטוט, כדי למנוע סרבול.



ברכותיי, כעת יש בידיכם את כל המידע הדרוש, כדי להבין את עבודת המעבד ב-2 תצורותיו - זו החד מחזורית, וזו המוצנרת.


Hazard Detection Unit
ID/EX.MemRead
EX.Flash
Stall
ID.Flash
Control
IF/ID.Write
IF.Flash
WB
MEM
EX
Cause
EPC
PC.Src
PC+4
Exception
Branch
Mux
PC
Instruction Memory
זיכרון הפקודות
ADD
4
Sign Extend
Shift left 2
IF/ID
ID/EX
Register File
מקבץ האוגרים
Write Address
Write Data
rs
rt
rd
rs-4
rs-5
rt-4
rt-5
ALU
EX/MEM
Data Memory
זיכרון הנתונים
MEM/WB
ALU Result
Dst of 3
Dst address of 4
Dst address of 5
Forwarding Unit
יחידת ההעברה קדימה
rs address
rt address