המעבד בתצורת חד מחזורי

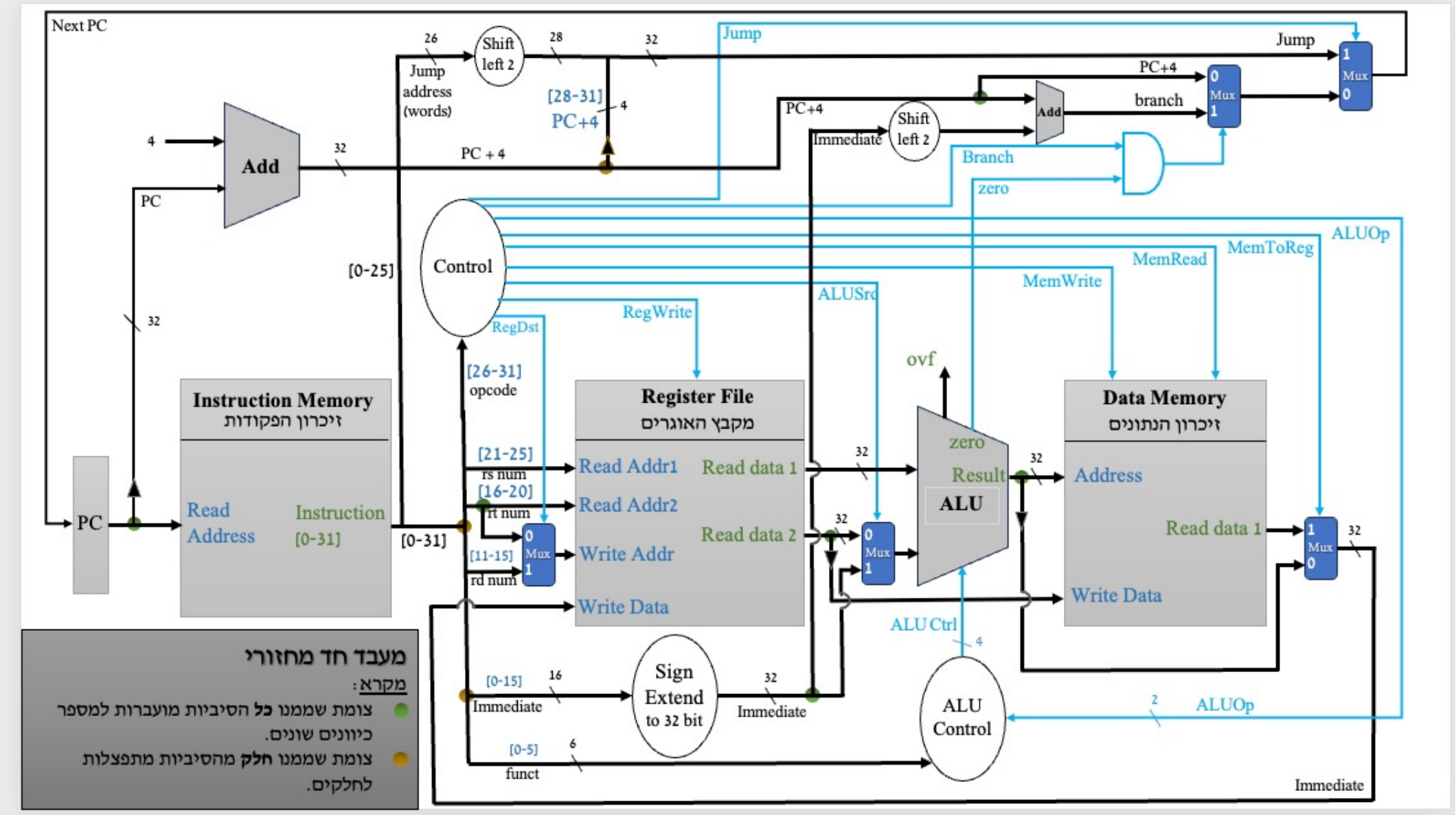
בשיעור זה נדבר על המעבד בתצורת חד מחזורי, ועל מסלולה של פקודה במעבד בתצורה זו.
בתצורה זו, בכל פעימת שעון עוברת פקודה אחת מסלול מלא במעבד מקצה לקצה.
אוסף הפקודות שאותן עלינו לבצע, נמצא ברכיב זיכרון שנקרא "זיכרון הפקודות"
בין היתר, הרכיב שאחראי להחזיק את הכתובת של המקום המתאים בזיכרון הפקודות שבו נמצאת הפקודה הבאה שברצוננו לקרוא - הוא אוגר שנקרא PC.
ברכיבים בהם אנחנו נעסוק, כל תא בזיכרון מכיל מידע בגודל של בית אחד - כלומר 8 סיביות, ולכן הכתובות בזיכרון הן ביחידות מידע של בתים.
פקודה באסמבלי - מיוצגת ע"י 32 סיביות, ולכן מכיוון שכל בית מכיל 8 סיביות, יש צורך ב 4 בתים בזיכרון כדי לשמור את התוכן של פקודה אחת.
לכן הכתובות אליהן אנחנו ניגשים בזיכרון - הן בכפולות של 4.
אנחנו מתחילים את התהליך כאשר האוגר PC מעביר 32 סיביות של כתובת שהוא מחזיק. 32 סיביות אלו - הן הייצוג בינארי של מספר, שהוא בעצם מספר של תא בזיכרון, שהחל ממנו אנחנו סופרים 4 תאים - כלומר 4 בתים - כלומר 32 סיביות, שזהו הגודל שנדרש כדי לשמור קידוד של פקודה אחת באסמבלי. (על קידוד פקודה תוכלו לשמוע בסרטון מספר 7)
אם כך בעצם מה שקורה הוא שהאוגר PC מעביר את הכתובת של הפקודה הבאה לרכיב "זיכרון הנתונים", רכיב זה ניגש ל-4 התאים הסמוכים החל מכתובת זו - שכזכור 4 תאים אלו מכילים את הקידוד של הפקודה הבאה,
ומוציא את קידוד הפקודה ב 32 סיביות הלאה. 32 סיביות אלו מתפצלות לפי שדות.

לפני שנמשיך - ניתן תזכורת קטנה:
לכל פקודה יש שדות מסוימים כגון RS , RT, opcode , imidiate וכדומה..
כשאנחנו מדברים על השדות, הכוונה היא רק למספרי הסיביות שהם מייצגים , ולא על משמעותם בפועל,
לכן נוכל למשל לדבר על שדה ה RD בפקודה מסוג I, למרות שלשדה זה תפקיד מיוחד בסוג זה של פקודות,
כשנעשה שימוש שכזה, המשמעות תהיה לדבר על הסיביות בפקודה, שנמצאות באינדקסים 11-15.
נזכיר שאת מספרי הסיביות מתחילים לספור מ 0 ועד 31 - סך הכל 32 סיביות.
להלן סוגי שדות שונים עליהם נדבר - ומספרי הסיביות שהם מייצגים:

- opcode [26-31] - 6 bits
- rs [21-25] - 5 bits
- rt [16-20] - 5 bits
- rd [11-15] - 5 bits
- shamt [6-10] - 5 bits
- funct [0-5] - 6 bits
- immediate [0-15] - 16 bits
- Jump [0-25] - 26 bits

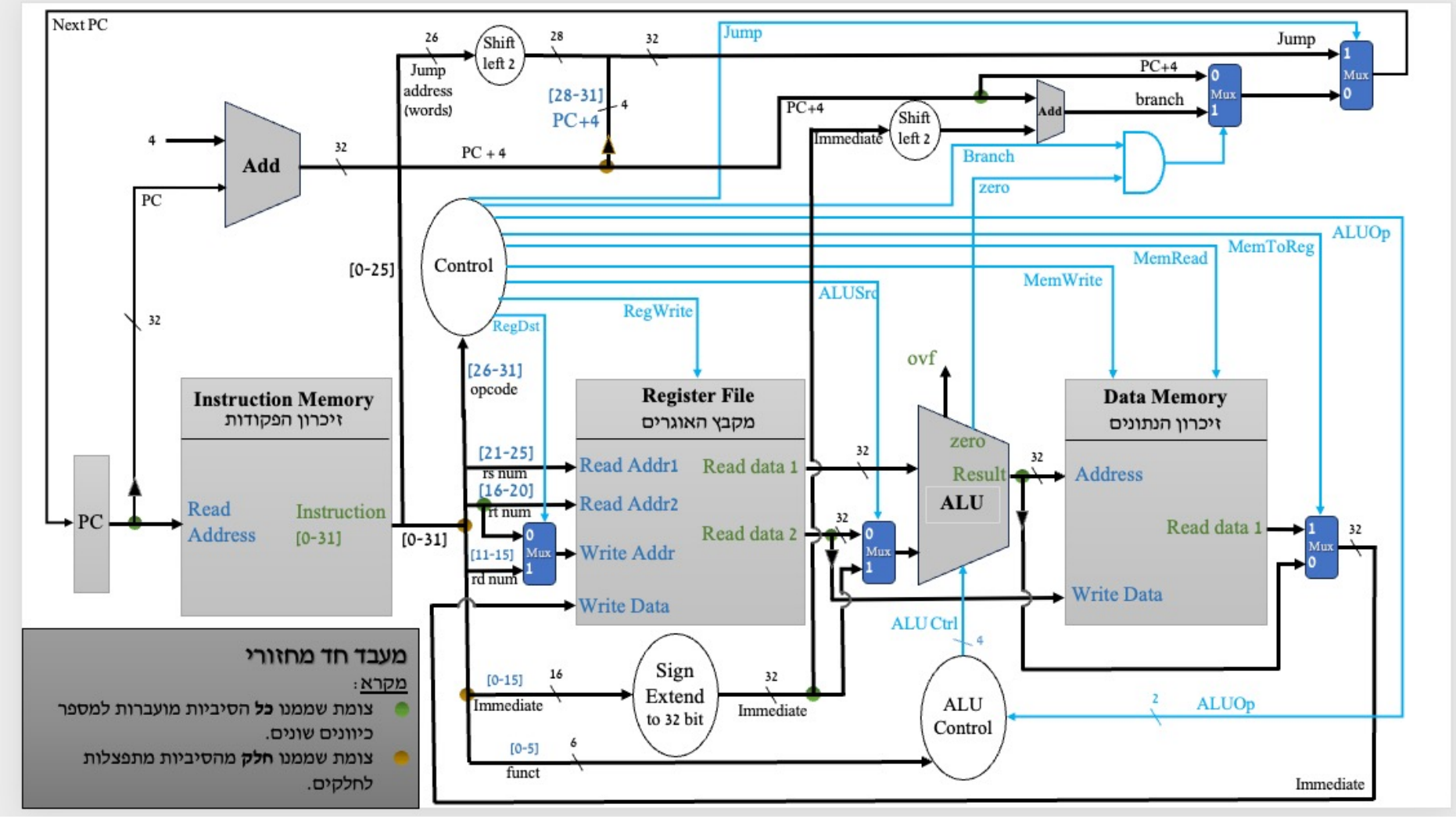
32 הסיביות של קידוד הפקודה, מתפצלות לשדות הבאים:

- שדה ה opcode (סיביות 26-31) נכנס לרכיב ה Control Unit , בהמשך נרחיב על תפקידו, אך כעת נסתפק בלומר שהוא אחראי לודא שהפקודה מתבצעת בצורה נכונה ברכיבים השונים, למשל שתהיה קריאה מהזיכרון רק בפקודות מסוג LW, או שרכיב ה ALU יבצע חיבור רק כשמדובר על פקודת add וכדומה..
  משמעותו של שדה זה - להעיד על סוג הפקודה - כלומר למשל מסוג R , מסוג I או מסוג jump.



- שדה ה **rs** (סיביות 21-25) מכיל מספר אוגר, ונכנס למקבץ האוגרים לכניסה Read Addr 1. אם נניח למשל שהסיביות שמכיל שדה זה הן 01100 (כלומר 12) המשמעות תהיה שהאוגר הראשון שיעניין אותנו - הוא אוגר מספר 12.
- שדה ה **rt** (סיביות 16-20) נכנס למקבץ האוגרים לכניסה שנקראת Read Addr 2. אם נניח למשל שהסיביות שמכיל שדה זה הן 10001 (כלומר 17) המשמעות תהיה שהאוגר השני שיעניין אותנו - הוא אוגר מספר 17.
- שדה ה **rd** (סיביות 11-15) נכנס למקבץ האוגרים לכניסה שנקראת Write Addr. מה תפקידו?

בפקודות מסוג R, שבהן צריכה להתבצע כתיבה לאוגר מסוים (כלומר שינוי המידע שהוא מכיל), המשמעות תהיה שנרצה לכתוב באוגר, איזה אוגר? האוגר שמספרו מקודד בשדה זה. למשל אם rd=00100, אזי נרצה לכתוב באוגר מספר 4.



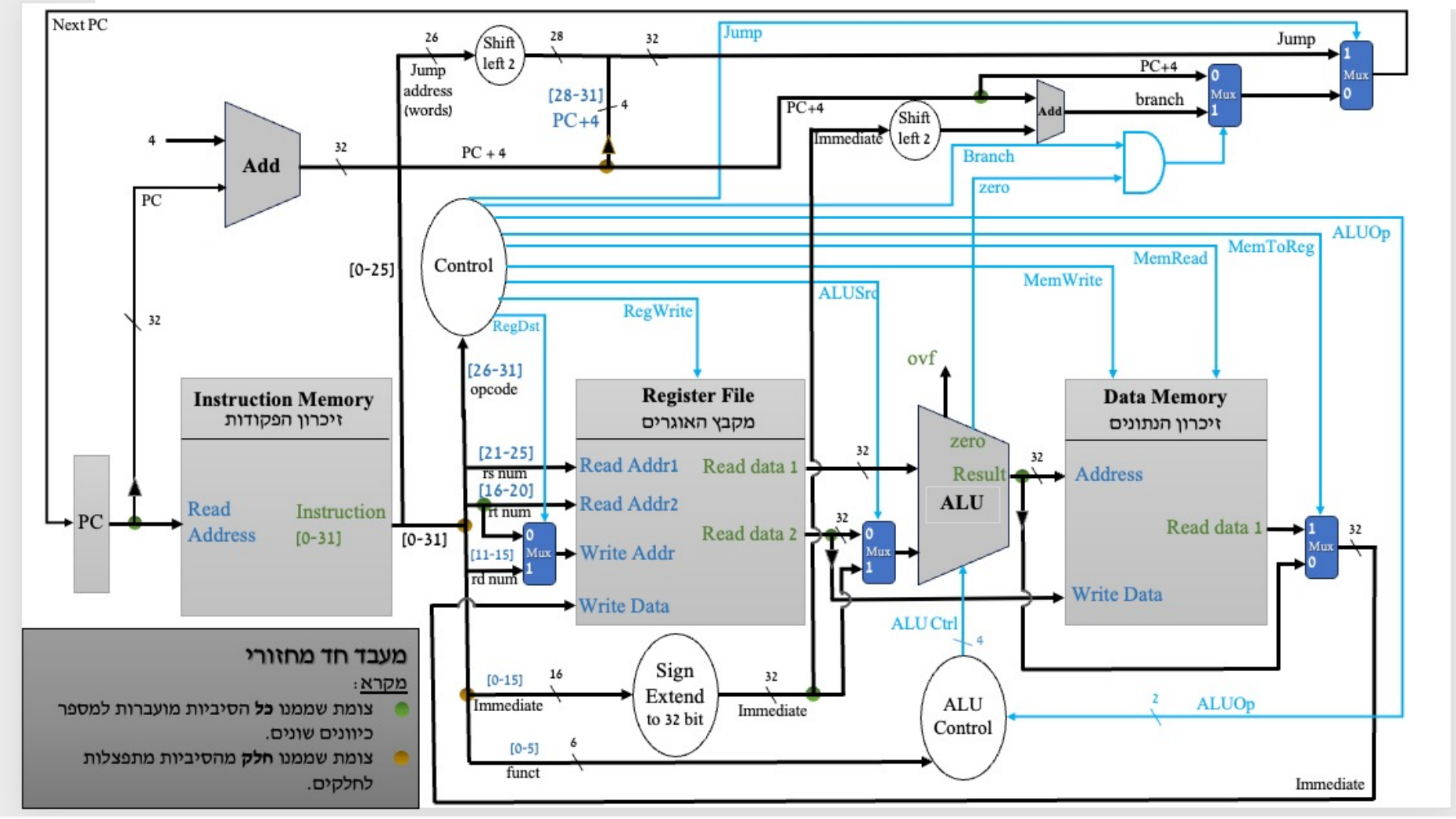
- שדה ה **immediate** (סיביות 0-15) מייצגות מספר ממש, עבור פקודות בהן צריך לעשות פעולה עם מספר קבוע, למשל אם נרצה לקחת את המספר ששמור בתוך אוגר מספר 3 ולהוסיף לו את הערך 13, אזי 13 יהיה הערך המיידי).

נשים לב לעניין הבא: בפקודות מסוג I, יש שימוש בשדות opcode,RT,RS שמספר הסיביות בהן הם משתמשים, הוא סך הכל 16 (5+5+6), לכן עבור פעולות עם ערך מיידי - נותרו רק 16 סיביות פנויות.

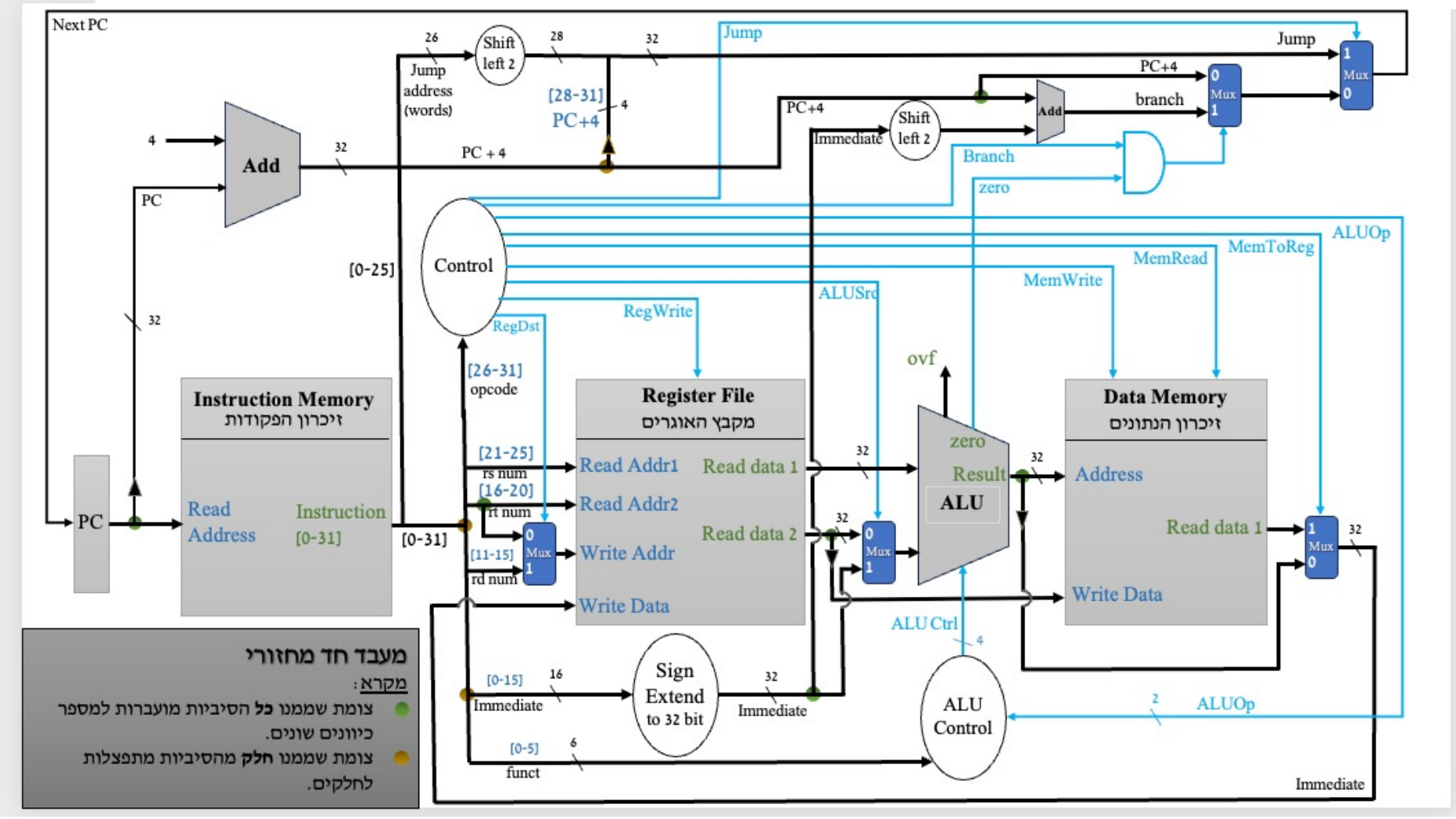
אולם רכיבי המעבד, יודעים לבצע פעולות על מספרים/ערכים המיוצגים ב 32 סיביות. לכן עלינו קודם כל להמיר את הערך המיידי ל 32 סיביות בשיטת המשלים ל-2.

הרחבה זו מ 16 סיביות ל 32 סיביות, נעשית באמצעות רכיב הנקרא sign extent והוא פועל כך:

- אם המספר היה חיובי, כל שצריך לעשות זה להוסיף אפסים מצד שמאל ב 16 סיביות.
- אם המספר היה שלילי, כדי לשמור על ערכו בשיטת המשלים ל 2, עלינו להוסיף **אחדים** מצד שמאל ב 16 סיביות.



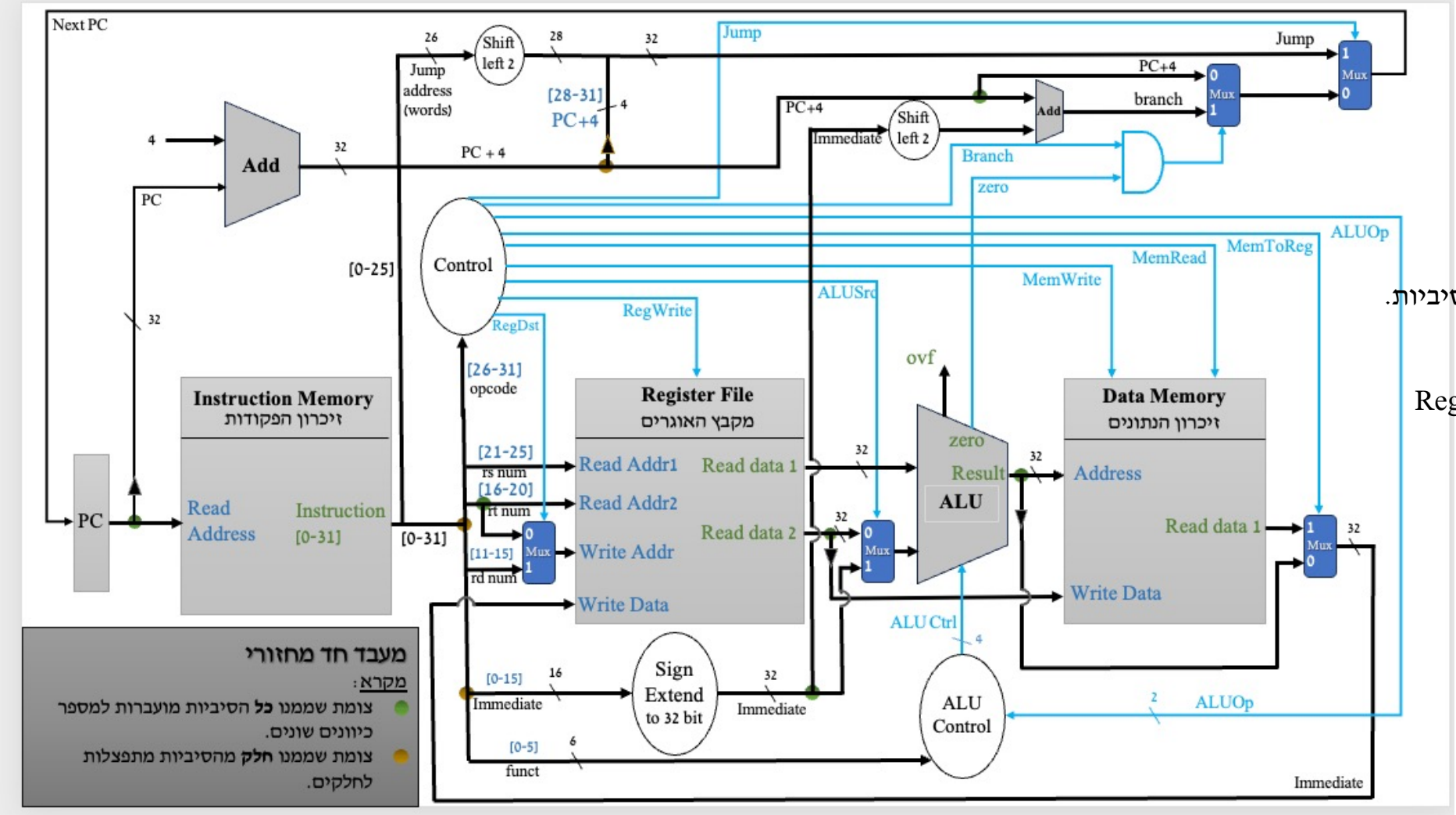
- שדה ה **funct** מעיד בפקודות מסוג R על סוג הפעולה שצריך לבצע על 2 האופרנדים שנכנסו ל ALU.
הבקרה הראשית מקבלת את ה opcode ומוציאה אל הבקרה המשנית ALU Control , פלט בקרה של 2 סיביות בשם ALU Op , הבקרה הראשית מקבלת גם את שדה ה funct, ובאמצעות פונקציה בוליאנית - הופכת את 8 הסיביות שקיבלה בסה"כ כקלט ל-4 סיביות פלט ALU Ctrl לרכיב ה ALU, בעזרתם הוא יודע איזה פעולה לבצע על 2 האופרנדים, למשל חיבור, חיסור, OR , AND וכו..
נשים לב שבפקודות מסוג I, איך חשיבות לשדה זה, והפלט תלוי מבחינה מהותית רק ב ALU Op שמגיע מהבקרה הראשית
- שדה ה **Jump** (סיביות 0-25) מייצג כתובת ביחידות מידה של <u>מילים</u>, נזכיר שקידודה של כל פקודה, מצריך מילה אחת גם כן (4 בתים). מכיוון שבמעבד יחידת המידע המקובלת לזיכרון היא בבתים, קודם כל מתבצעת המרה לבתים ע"י הכפלה ב-4, ומתקבלות 28 סיביות. את 4 הסיביות העליונות החסרות של הכתובת, לוקחים מ PC+4, כלומר הדבר מאפשר טווח קפיצה של 2^{26} <u>פקודות</u>)כלומר ניתן להגיע ל 64 מיליון פקודות ע"י פקודת Jump).



אם כן קידוד הפקודה שלנו יוצא מזיכרון הפקודות ב 32 סיביות, ומגיע בין היתר גם לרכיב מקבץ האוגרים.
מקבץ האוגרים הוא אוסף של 32 אוגרים - הממוספרים מ 0 ועד 31 בהם ניתן להשתמש לשימושים שונים.
כאשר נרצה לפנות לאוגר מסוים, נוכל "לקרוא" לו באמצעות המספר שלו.
כיוון שיש סך הכל 32 אוגרים, מספיקות 5 סיביות כדי למספר את כולם (שהרי 2 בחזקת 5 שווה 32)
למשל מספרו של אוגר מספר 13 ב 5 סיביות הוא 01101.
מספרו של אוגר מספר 31 ב 5 סיביות הוא 11111.
נפרט את הכניסות והיציאות של מקבץ האוגרים:

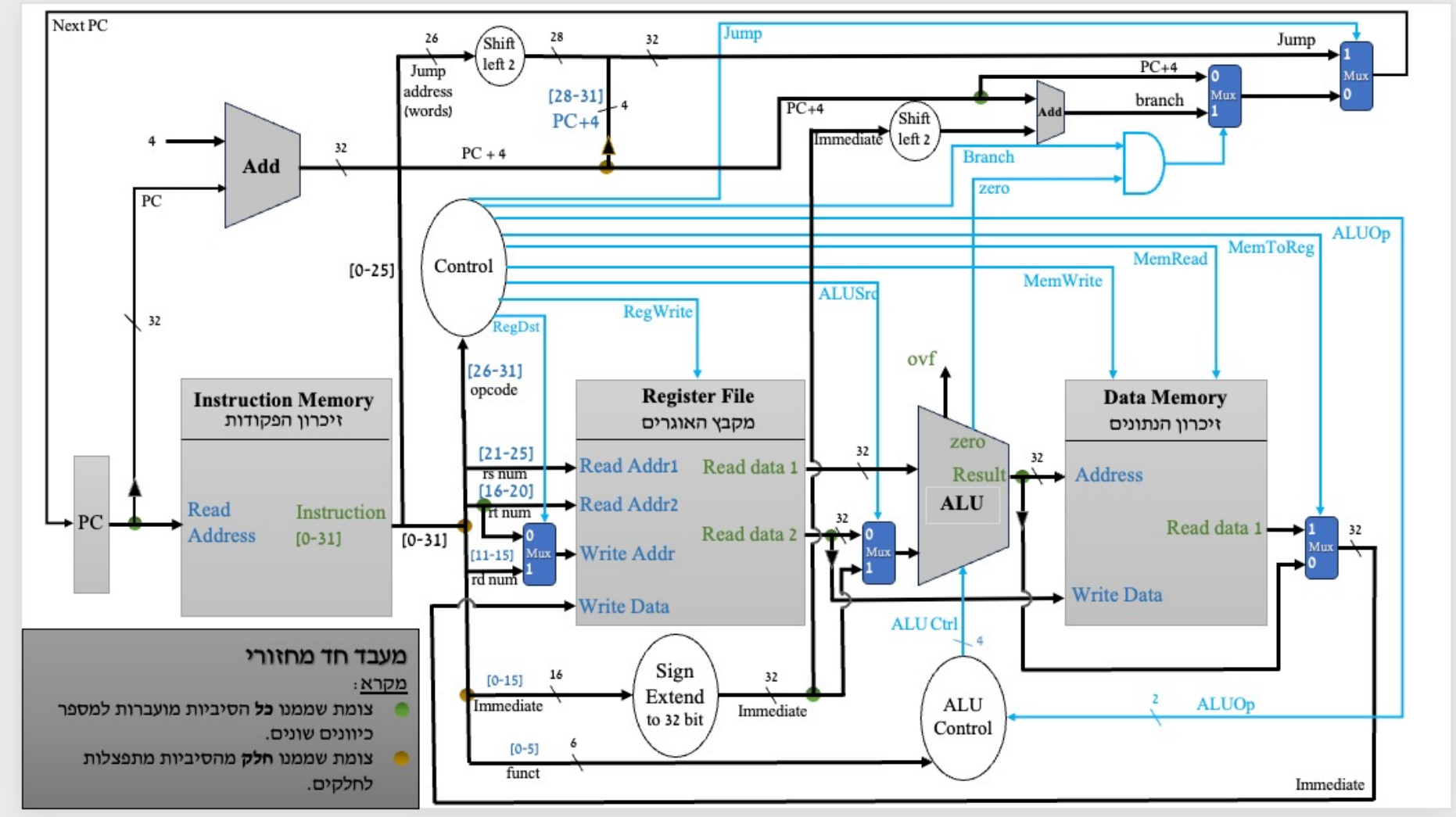
- הכניסה Read Address 1 תקבל את מספרו של האוגר rs ב 5 סיביות
- היציאה Read Data 1 תוציא את תוכן האוגר rs ב 32 סיביות
- הכניסה Read Address 2 תקבל את מספרו של האוגר rt ב 5 סיביות
- היציאה Read Data 2 תוציא את תוכן האוגר rt ב 32 סיביות
- הכניסה Write Address תקבל את מספרו של האוגר שנרצה לכתוב בו ב 5 סיביות.

נציין שמספרו של אוגר זה יכול להיות או מספרו של האוגר rt או מספרו של האוגר rd.
הבחירה נעשית ע"י מרבב, עם קו בקרה RegDst
(מהמילה destination שפירושה יעד,
כלומר מי הוא אוגר היעד שנרצה לכתוב בו).



❖ עולה השאלה, מה קורה אם זו בכלל פקודה שלא צריך לכתוב בה במקבץ האוגרים? לשם כך יש קו בקרה שמגיע למקבץ האוגרים ושמו RegWrite ורק כאשר הוא דולק (כלומר ערכו 1) תתבצע כתיבה במקבץ האוגרים.
בפקודות שבהן לא אמורה להתבצע כתיבה למקבץ האוגרים, קו בקרה זה יהיה כבוי (כלומר ערכו 0).

- הכניסה WriteData תקבל את התוכן ב-32 סיביות שנרצה לכתוב באוגר שאת מספרו קיבלנו ב Write Address.



ה ALU:

רכיב זה נועד לבצע פעולות חישוביות על 2 אופרנדים שכל אחד מהם הוא ב 32 סיביות

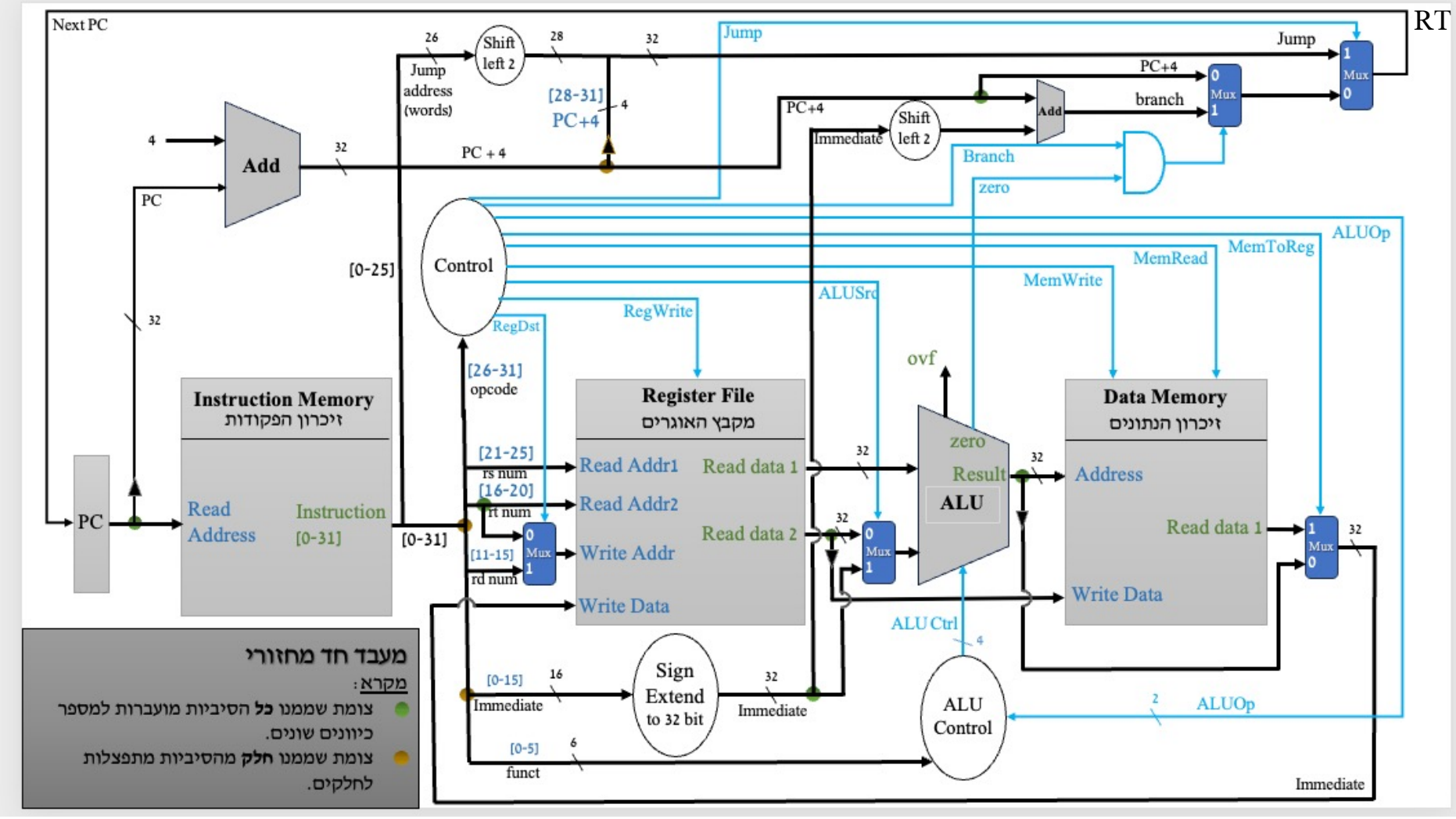
המשמעות של המילה אופרנד - היא בעצם הדבר שעליו פועל האופרטור.

לדוגמה נתבונן בתרגיל החיבור :2+3=5בתרגיל זה האופרטור הוא פלוס, שהרי שהפעולה שאנחנו מבצעים היא פעולת חיבור.

האופרנד הראשון כלומר הדבר הראשון שעליו פועלת פעולת החיבור (אופרטור החיבור) הוא המספר 2. האופרנד השני שעליו פועל האופרטור הוא 3.

הכניסות והיציאות שלו ה ALU:

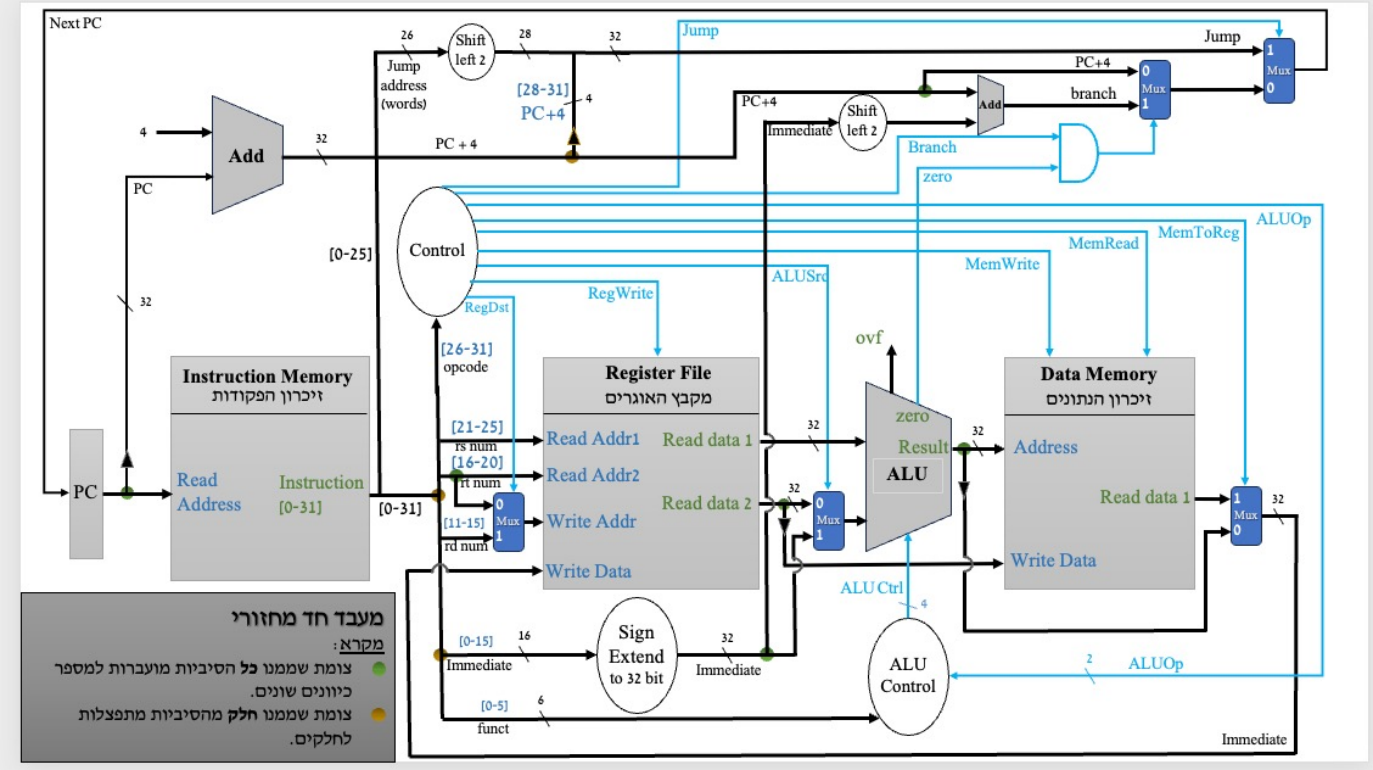
- בכניסה העליונה, נכנס האופרנד הראשון, והוא מגיע ישירות מ Read Data 1 - כלומר האוגר RS, ב 32 סיביות.
- בכניסה התחתונה, יש 2 אפשרויות:
  - או שהאופרנד השני יגיע מהאוגר RT
  - או שהאופרנד השני הוא ערך מיידי כלומר מספר ממש, שהגיע משדה ה immediate, לאחר שעבר הרחבה ל 32 סיביות.



כדי לאפשר בחירה בין 2 האפשרויות, נוסף מרבב **mux**, שהיציאה שלו, נכנסת לכניסה התחתונה ב **ALU**.
הבחירה במרבב בין 2 האפשרויות תֵּעָשֶׂה באמצעות קו בקרה, שנקרא ALU Src ומגיע מרכיב הבקרה Control Unit.
ואיך ה control unit יודע איזה אופרטור צריך לקחת? הוא עושה זאת באמצעות שדה ה opcode שנכנס אליו.
כך למשל הבקרה תזהה פקודות מסוג R, בזכות ה opcode שלהן, שהוא תמיד 0. בפקודה אלה, אין שימוש בשדה הערך המיידי, ולכן הכניסה שתיבחר עבורן במרבב (Mux) תמיד תהיה זו שמגיעה ממקבץ האוגרים (0).
לעומת זאת בפקודות <u>מסוימות</u> מסוג I, בהן יש שימוש בשדה המיידי, הכניסה שתיבחר במרבב תהיה של הערך המיידי (1)
לאחר שעבר הרחבה כמו שהוסבר קודם.

- כניסת הבקרה ALU Ctrl מגיעה מהבקרה המשנית ב 4 סיביות ותפקידה לומר ל ALU איזה פעולה לבצע על 2 האופרנדים.
- היציאה result תכיל את התוצאה ב 32 סיביות.
- היציאה zero היא סיבית אחת שמקבלת ערך '1' רק כאשר 2 האופרנדים שווים, ניתן לראות שימוש בה, למשל בפקודת branch if equal.
- היציאה ovf - קיצור של overflow , מדובר בסיבית אחת שמציינת האם יש גלישה, כלומר כאשר אין מספיק סיביות לייצג את התשובה.

לדוגמה נניח שנרצה לחבר בשיטת המשלים ל-2 את המספרים 01_b עם 01_b התוצאה אמורה להיות, $10_b = 2_{10}$
אך לצערנו בשיטת המשלים ל-2 כשיש 2 סיביות - הערך 10_b שווה ערך ל $(-2)_{10}$ ולכן תתקבל טעות. לייצוג התשובה בצורה נכונה בשיטת המשלים ל-2 אנחנו צריכים לפחות 3 סיביות, ואז יכולנו לרשום 010 ולא הייתה בעיה, מכיוון שיש רק 2 סיביות מתקבלת טעות ולכן זו גלישה.



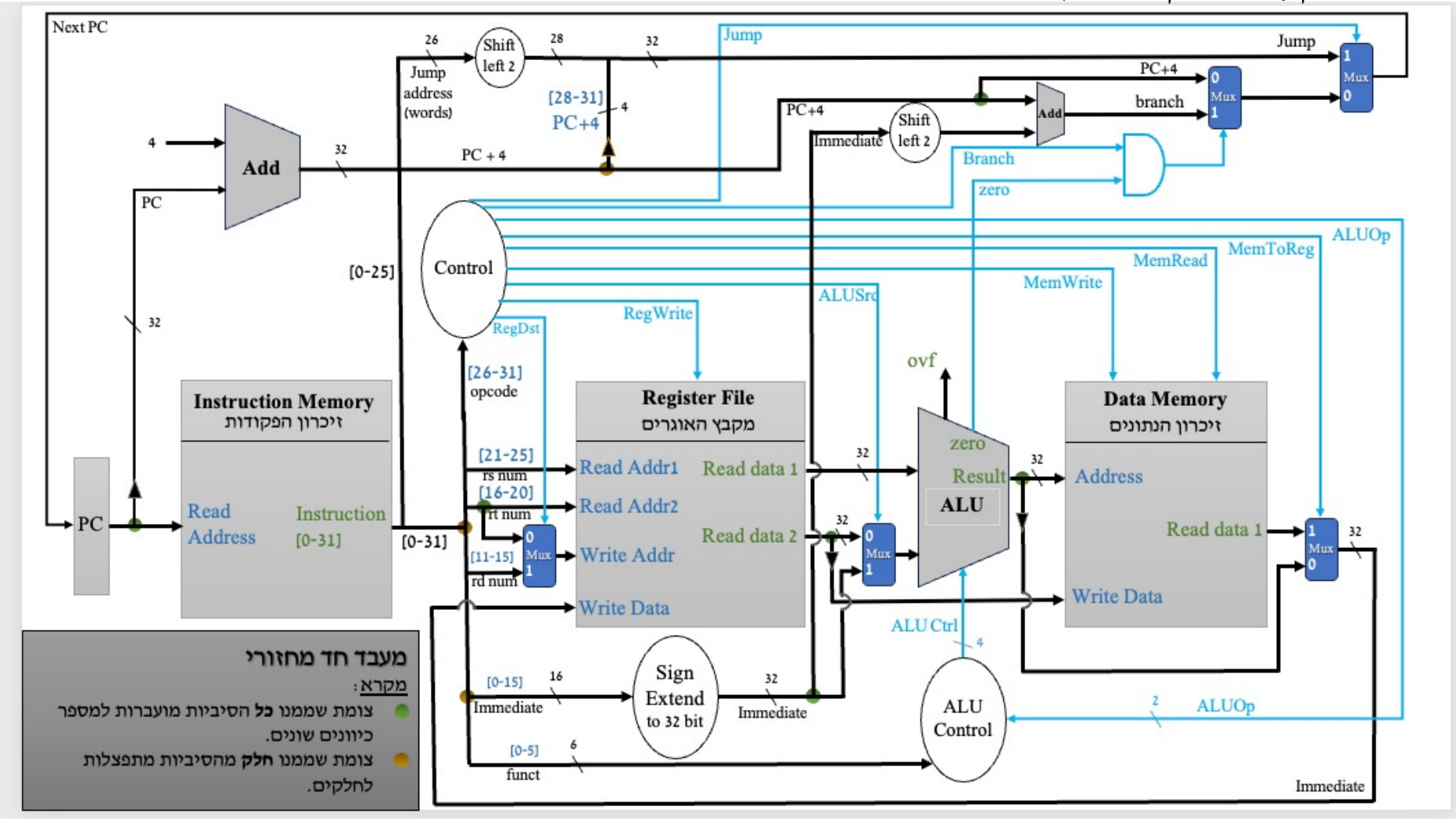
זיכרון הנתונים:

- הכניסה Address מייצגת כתובת ב 32 סיביות שחושבה ברכיב ה ALU, דבר זה נדרש בפקודות SW,LW בהן מחשבים את הכתובת בזיכרון שאליה צריך לגשת, החישוב הוא על ידי סכום של הכתובת שמאוחסנת בתוך האוגר rs עם ערך ההיסט (שהוא ערך מיידי).
- הכניסה Write Data מכילה את התוכן ב 32 סיביות שצריך לכתוב בזיכרון (אם בכלל) ומגיע, מאוגר rt ולכן יש בה צורך בפקודות SW.

כיצד נדע האם בכלל צריך לקרוא מהזיכרון והאם בכלל צריך לכתוב בזיכרון?

לשים כך יש 2 קווי בקרות שמגיעים מהבקרה הראשית, (שכזכור יודעת להעביר תוכן נכון לבקרות באמצעות ה opcode של כל פקודה):

- הבקרה MemRead דולקת רק אם צריך לקרוא מהזיכרון (כלומר בפקודות LW).
- הבקרה MemWrite דולקת רק אם צריך לכתוב בזיכרון (כלומר בפקודות SW).



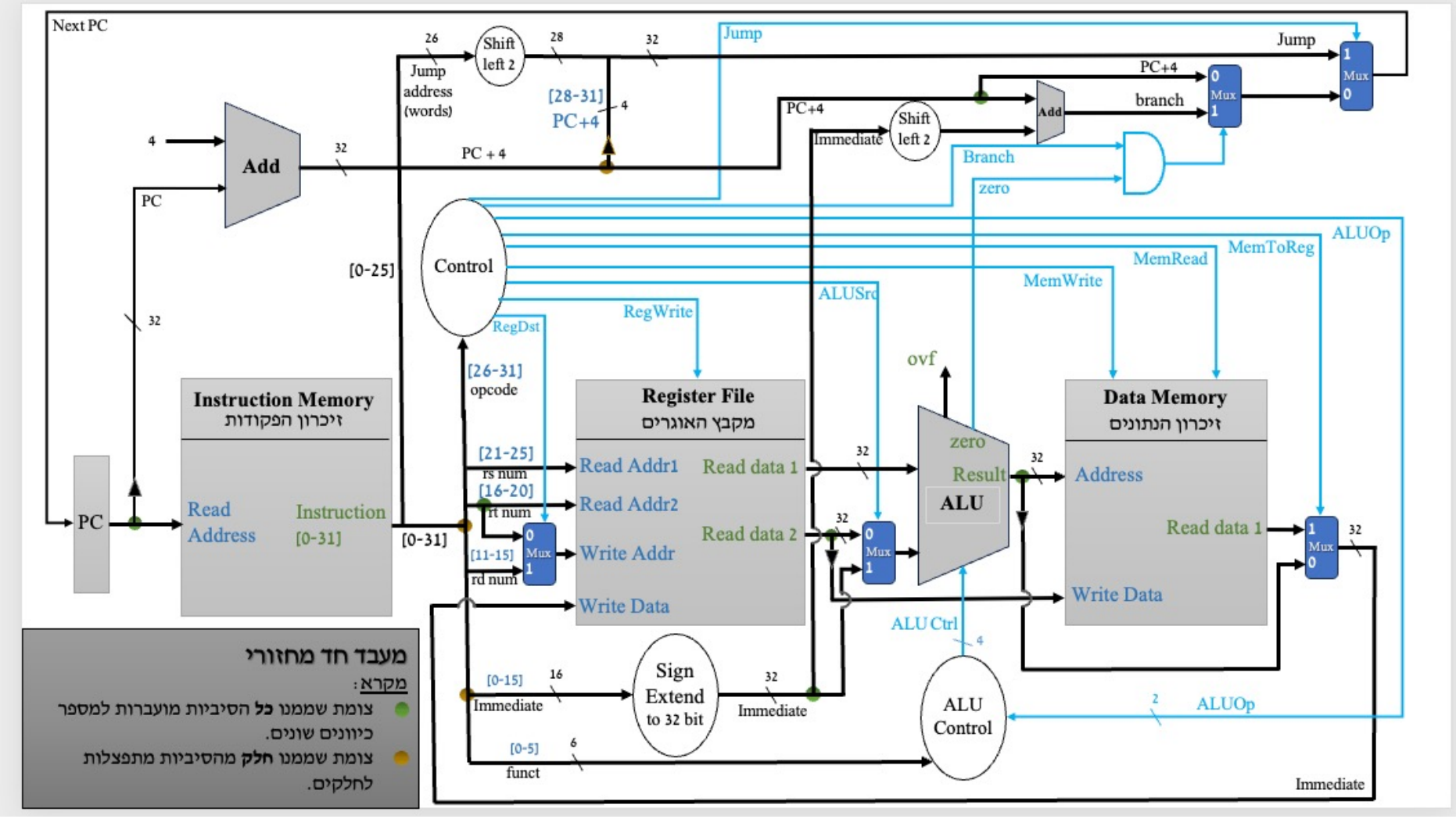
שלב ה Write Back (מיוצג ע"י המרבב שסמוך לזיכרון הנתונים)

בשלב זה צריך להיקבע הערך ב 32 סיביות שייכתב במקבץ האוגרים (בכניסה Write Data כמו שהוסבר קודם לכן), וזאת כמובן אם בכלל צריכה להתבצע כתיבה. קיימות 2 אפשרויות:

- ❖ אם הערך שצריך להיכתב במקבץ האוגרים, מגיע מזיכרון הנתונים (בפקודות LW) , עבור מצב זה קיימת הכניסה העליונה '1' במרבב.
- ❖ אם הערך שצריך להיכתב במקבץ האוגרים, חושב ב ALU (למשל בפקודות מסוג R) , עבור מצב זה קיימת הכניסה התחתונה '0' במרבב.

כיצד ידע המרבב באיזו אפשרות לבחור?

לשם כך מגיע אליו קו הבקרה MemToReg שמגיע מהבקרה הראשית.

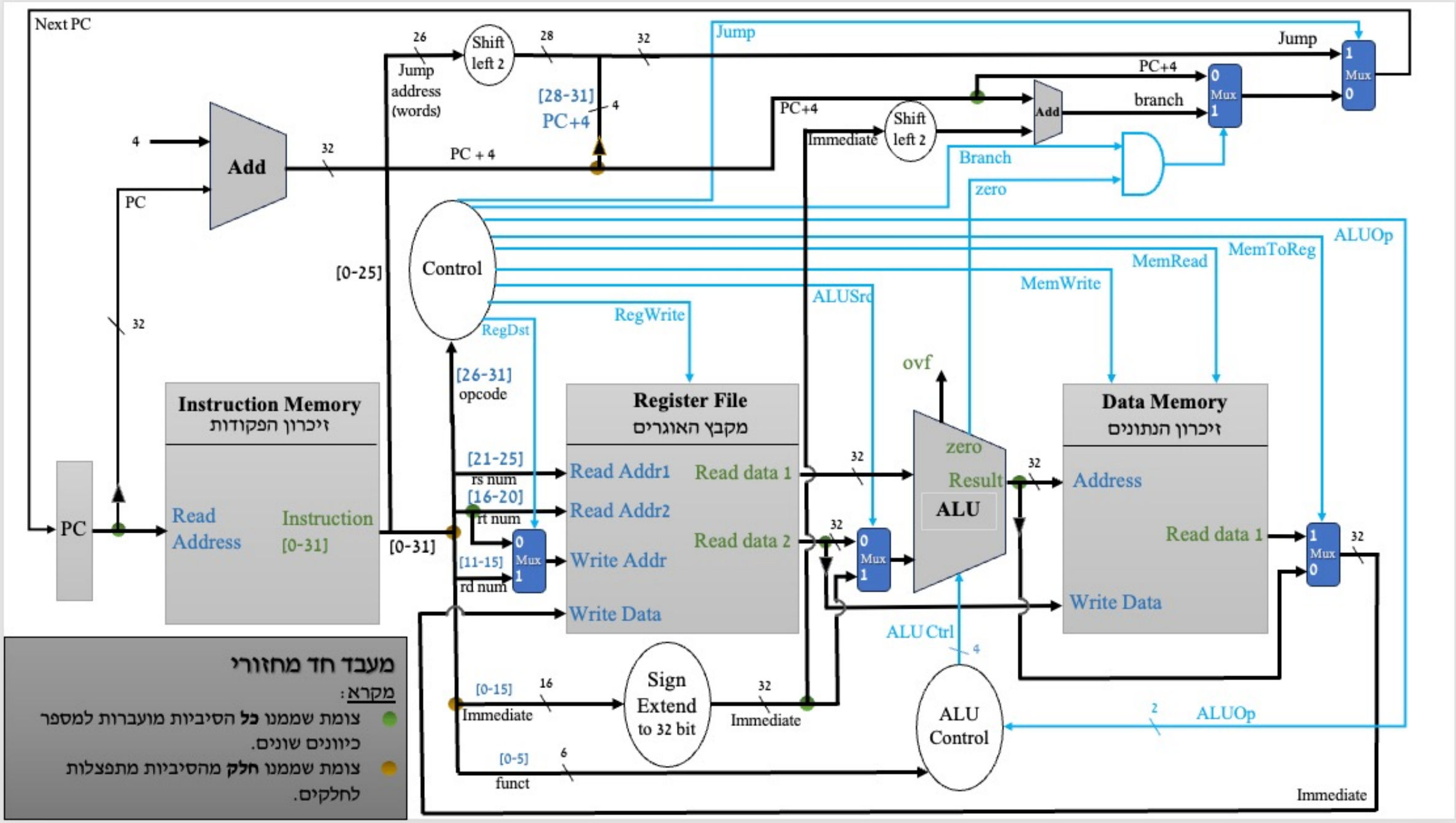


כעת נוכל לדבר על פקודות הקפיצה השונות בפרט, ועל חישוב הכתובת הבאה בכלל.
מי שאחראי על הכנסת הכתובת של הפקודה הבאה הוא האוגר PC.
ראשית נציין שקיימות 3 אפשרויות:

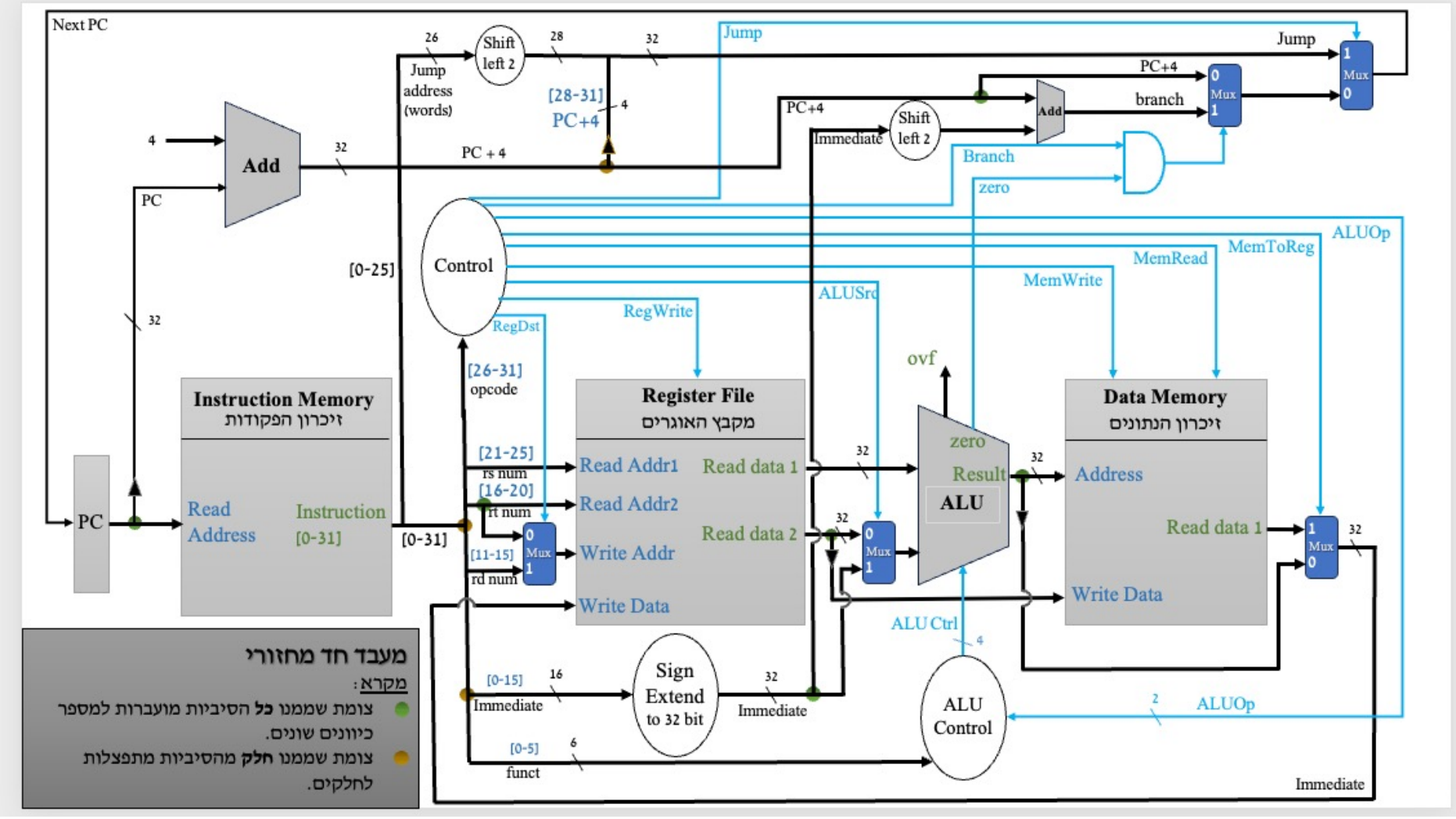
- התקדמות רגילה לפקודה הבאה, כלומר האוגר PC גדל ב-4 , המשמעות היא התקדמות רציפה ב-4 בתים מהכתובת הקודמת
כזכור כל פקודה תופסת בזיכרון הפקודות 4 בתים (שהם מילה אחת).
- קפיצה מסוג branch, שבה לוקחים כתובת הפקודה הבאה PC+4 ועליו מוסיפים את היסט הקפיצה שהוא ביחידות מידה של מילים,
היסט זה מגיע במקור מהערך המיידי, לאחר שעבר הרחבה ל-32 סיביות, אולם מכיוון שההיסט הוא ביחידות מידה של מילים (כלומר 4 בתים)
אז אנחנו צריכים להכפיל את הערך שהתקבל פי 4 (הדבר מתבצע ע"י הזזה שמאלה של 2 סיביות ברכיב shift left 2).
- קפיצה מסוג Jump, בפקודה מסוג זה, ניתן לקפוץ רק לכתובות שבהן 4 הסיביות העליונות, הן 4 הסיביות העליונות של PC+4.
לכן 4 סיביות אלו נלקחות תחילה.
נשים לב שכדי לזהות פקודה מסוג jump יש צורך ב opcode שמצריך 6 סיביות, ולכן נותרות 26 סיביות פנויות, שהן שדה ה Jump.
26 סיביות אלו, הן ביחידות מידה של מילים, ולכן הן עוברות הרחבה ל 28 סיביות ע"י הוספת 2 אפסים מימין,כדי להגיע ליחידות מידה של בתים.
לבסוף מאחדים את 4 הסיביות העליונות של PC+4 עם 28 הסיביות הנמוכות שהגיעו משדה ה Jump.

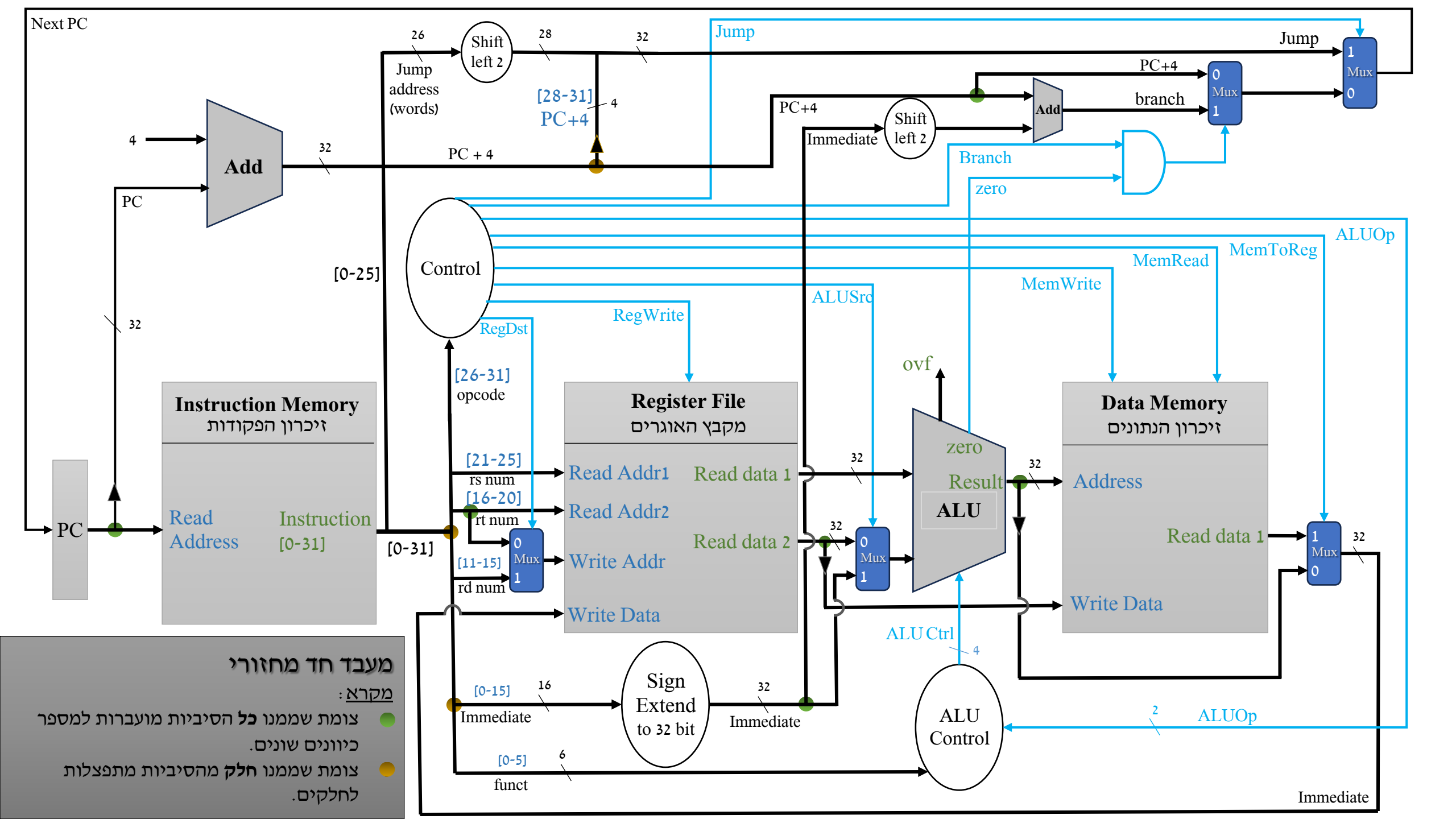
❖ מכאן נובע שהפקודות אליהן ניתן לקפוץ בפקודה מסוג Jump הן רק באזור בזיכרון שבו 4 הסיביות העליונות של הכתובת הן כמו של PC+4 .
אזור זה כולל בתוכו 2^{26} פקודות ו- 2^{28} בתים.

❑ אם למשל הכתובת של PC+4 היא 0xAB67FFD0 אזי 4 הסיביות הבינאריות העליונות מיוצגות ע"י ספרת ההקסא A.
מכאן נובע שהכתובות אליהן ניתן להגיע בפקודת Jump במיקום זה, יהיו החל מהכתובת 0xA0000000 שתתקבל עבור ערך בשדה jump=0x0,
ועד הכתובת 0xAFFFFFFC, שתתקבל עבור הערך בשדה jump=0x3FFFFFF=11 ... 111_b, נדגים כתובת זו:
לצורך המחשה נניח שהפקודה היא j 0x3FFFFFF , כזכור ערך זה מייצג 26 סיביות, לאחר שנרחיב ל-28 סיביות ע"י הוספת 2 אפסים מימין נקבל
את 0xFFFFFFC , עליו נוסיף את 4 הסיביות העליונות של PC+4 המיוצגות ע"י 0xA ונקבל 0xAFFFFFFC.
(כזכור בגלל הוספת האפסים מימין, שתי הסיביות הימניות הן בהכרח 0 ולכן ספרת ההקסא הימנית המירבית היא C ששקולה בבינארית ל 1100).


Next PC
26
Jump address (words)
Shift left 2
28
32
Jump
Jump
1 Mux 0
PC+4
0 Mux 1
branch
Add
PC+4
Shift left 2
Immediate
[28-31]
PC+4
4
4
Add
32
PC + 4
Branch
zero
PC
Control
[0-25]
ALUOp
MemToReg
MemRead
MemWrite
ALUSrc
RegWrite
RegDst
32
ovf
[26-31]
opcode
Instruction Memory
זיכרון הפקודות
Register File
מקבץ האוגרים
Data Memory
זיכרון הנתונים
[21-25]
rs num
[16-20]
rt num
[11-15]
rd num
Read Addr1
Read Addr2
Write Addr
Write Data
Read data 1
Read data 2
zero
Result
ALU
Address
Read data 1
Write Data
PC
Read Address
Instruction [0-31]
[0-31]
0 Mux 1
0 Mux 1
1 Mux 0
32
ALU Ctrl
4
[0-15]
16
Immediate
Sign Extend to 32 bit
32
Immediate
ALU Control
2
ALUOp
[0-5]
6
funct
Immediate
מעבד חד מחזורי
מקרא:
צומת שממנו כל הסיביות מועברות למספר כיוונים שונים.
צומת שממנו חלק מהסיביות מתפצלות לחלקים.

נותרה לנו סוגיה אחת אחרונה - כיצד בוחרים מי תהיה הכתובת הבאה. בודאי ניחשתם נכונה שגם כאן הבקרה הראשית מעורבת:
תחילה מתבצעת בחירה בין הערך PC+4 לבין הערך שחושב עבור פקודת beq, באמצעות מרבב עם קו בקרה PCSrc. כדי שתיבחר הפקודה
ה beq צריכים להתקיים 2 תנאים: א' - שיהיה בכלל מדובר בפקודת beq (הבקרה תדע בעזרת ה opcode). ב'- השוויון של פקודת ה beq,
כדי לדעת האם מתקיים שוויון בין 2 האופרנדים, משתמשים ביציאת ה zero של ה ALU שמעידה על כך שההפרש בינהם הוא 0 - כלומר הם שווים.
לאחר מכן מתבצעת בחירה במרבב נוסף בין הכתובת שנבחרה במרבב הקודם, לבין כתובת ה Jump שחושבה, באמצעות קו הבקרה Jump.
כעת - הכתובת שיוצאת ממרבב זה, תיכנס לאוגר PC שיזרים אותה מחדש לזיכרון הפקודות כדי שהיא תיקרא כפקודה הבאה.
ברכותי!
כעת יש בידיכם את כל המידע הדרוש להבנת פעילות המעבד החד מחזורי.




Next PC
Jump
Jump
Shift left 2
26
28
32
Jump address (words)
[28-31]
PC+4
4
PC+4
PC+4
branch
Add
Shift left 2
Immediate
Branch
zero
4
Add
32
PC + 4
PC
Control
[0-25]
ALUOp
MemToReg
MemRead
MemWrite
ALUSrc
RegWrite
RegDst
32
[26-31]
opcode
ovf
Instruction Memory
זיכרון הפקודות
Register File
מקבץ האוגרים
Data Memory
זיכרון הנתונים
[21-25]
rs num
[16-20]
rt num
[11-15]
rd num
Read Addr1
Read Addr2
Write Addr
Write Data
Read data 1
Read data 2
zero
Result
ALU
Address
Read data 1
Write Data
PC
Read Address
Instruction [0-31]
[0-31]
ALU Ctrl
4
[0-15]
16
Immediate
Sign Extend to 32 bit
32
Immediate
ALU Control
2
ALUOp
[0-5]
6
funct
Immediate
מעבד חד מחזורי
מקרא:
צומת שממנו כל הסיביות מועברות למספר כיוונים שונים.
צומת שממנו חלק מהסיביות מתפצלות לחלקים.