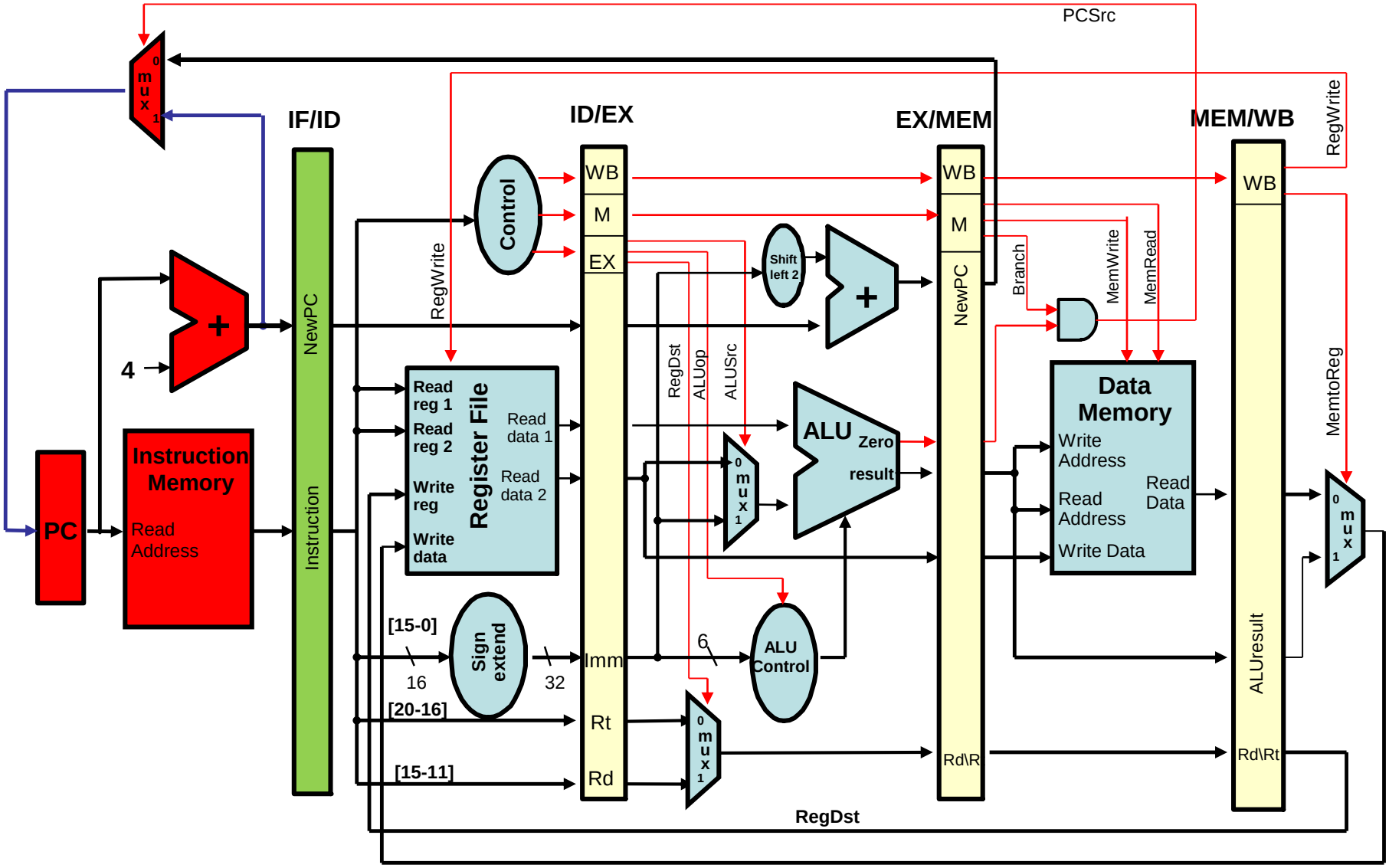
מבנה מחשבים ספרתיים
234267

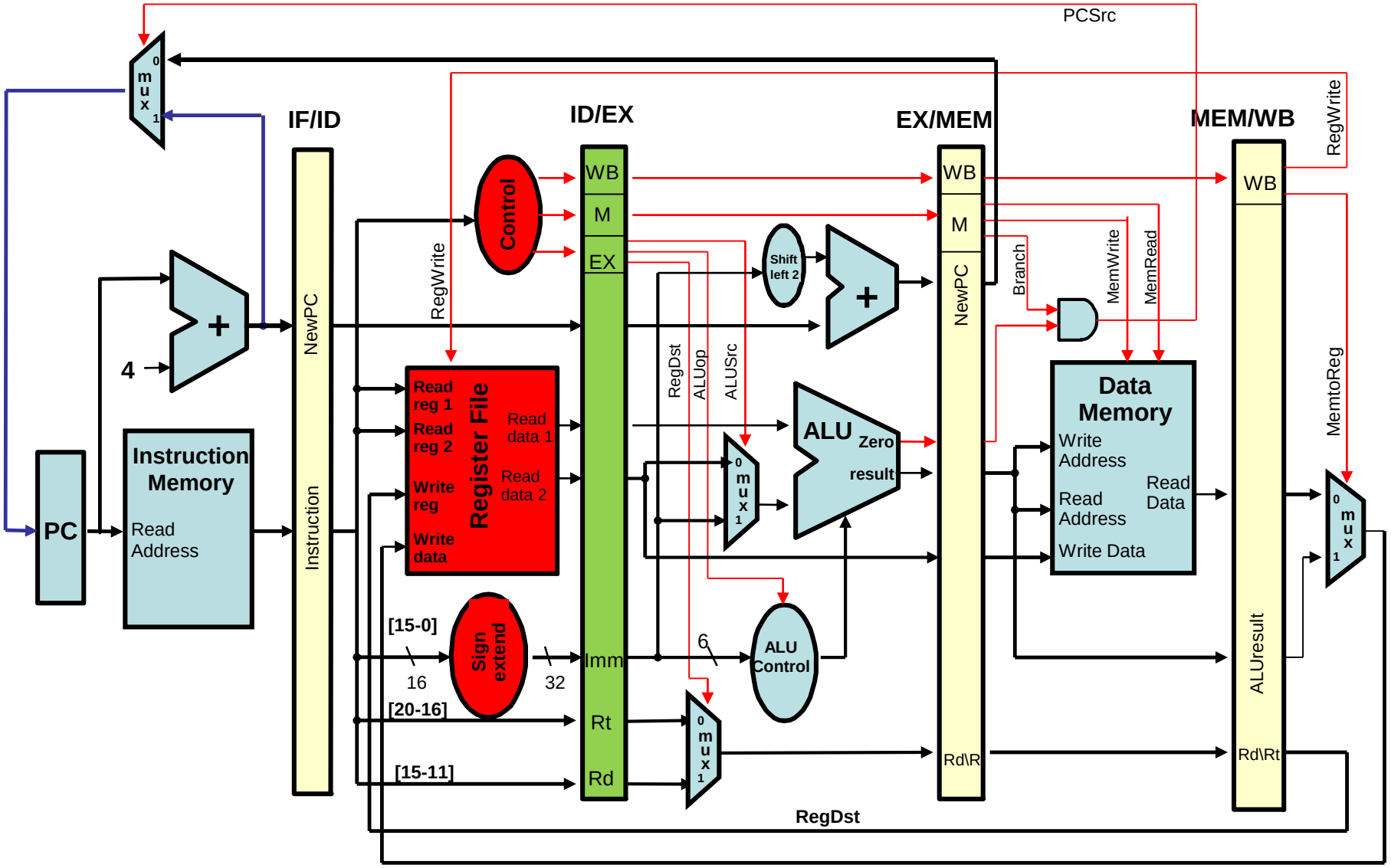
תרגול מס' 2:

Data and Control Hazards

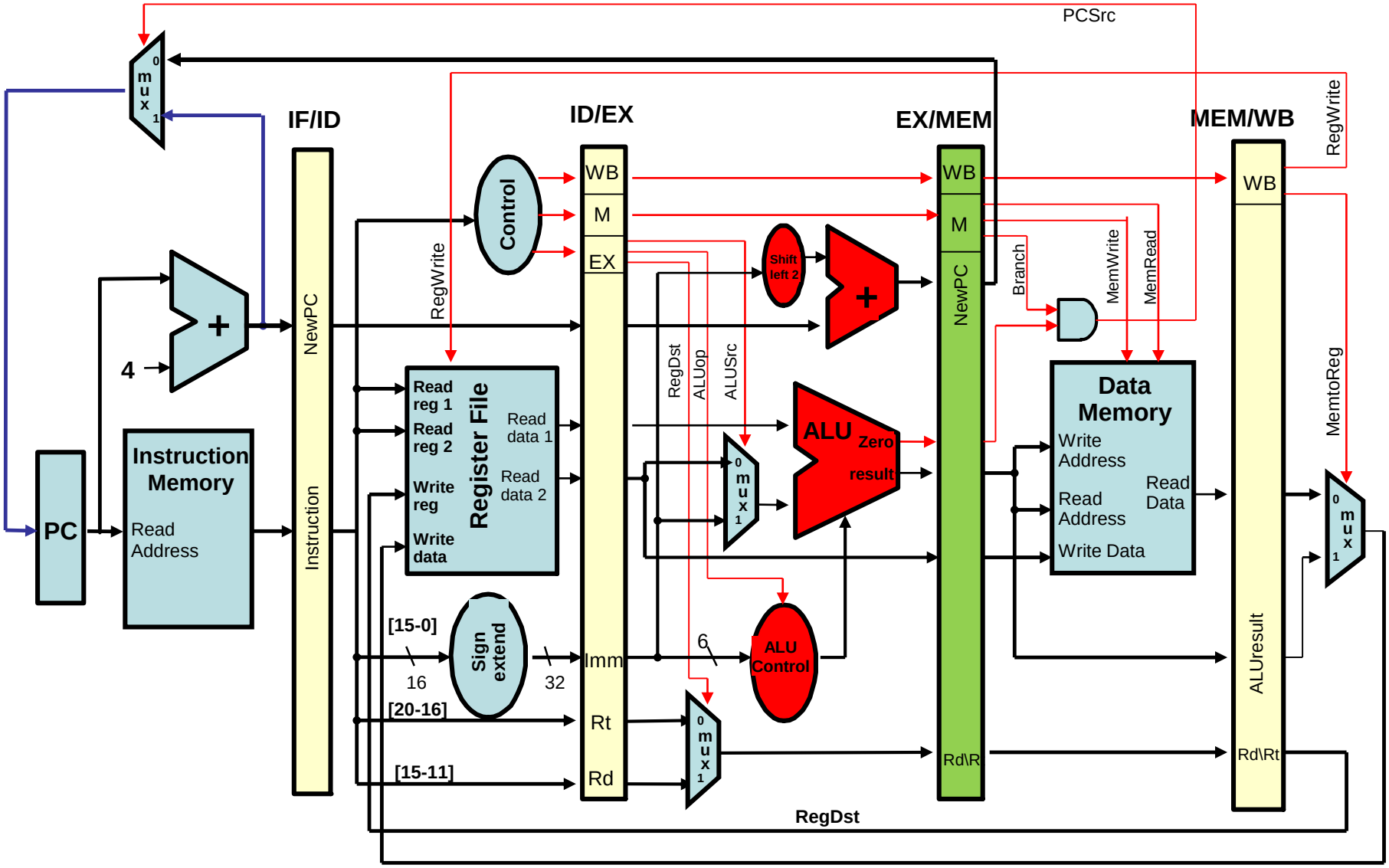
Instruction Fetch



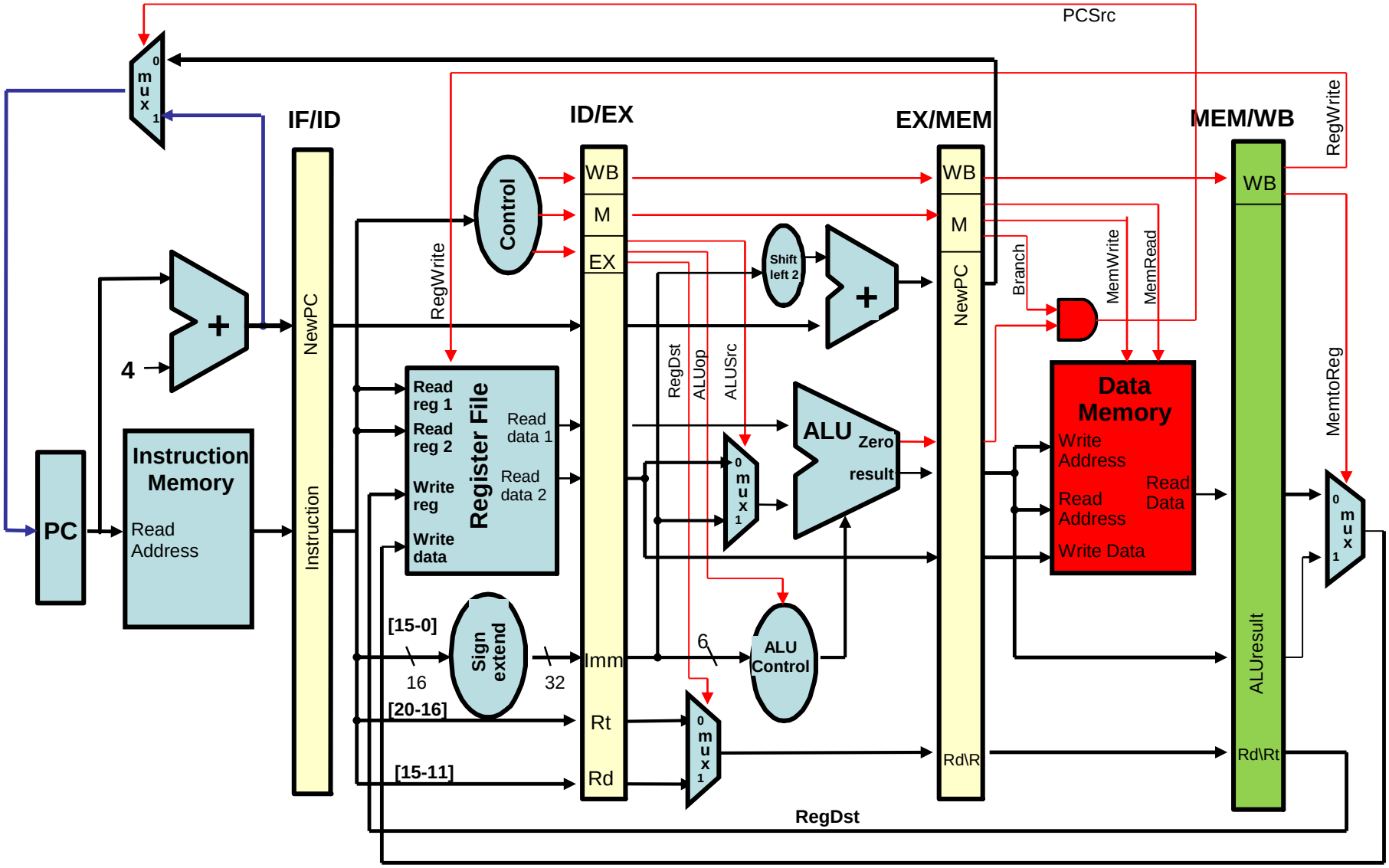
Instruction Decode



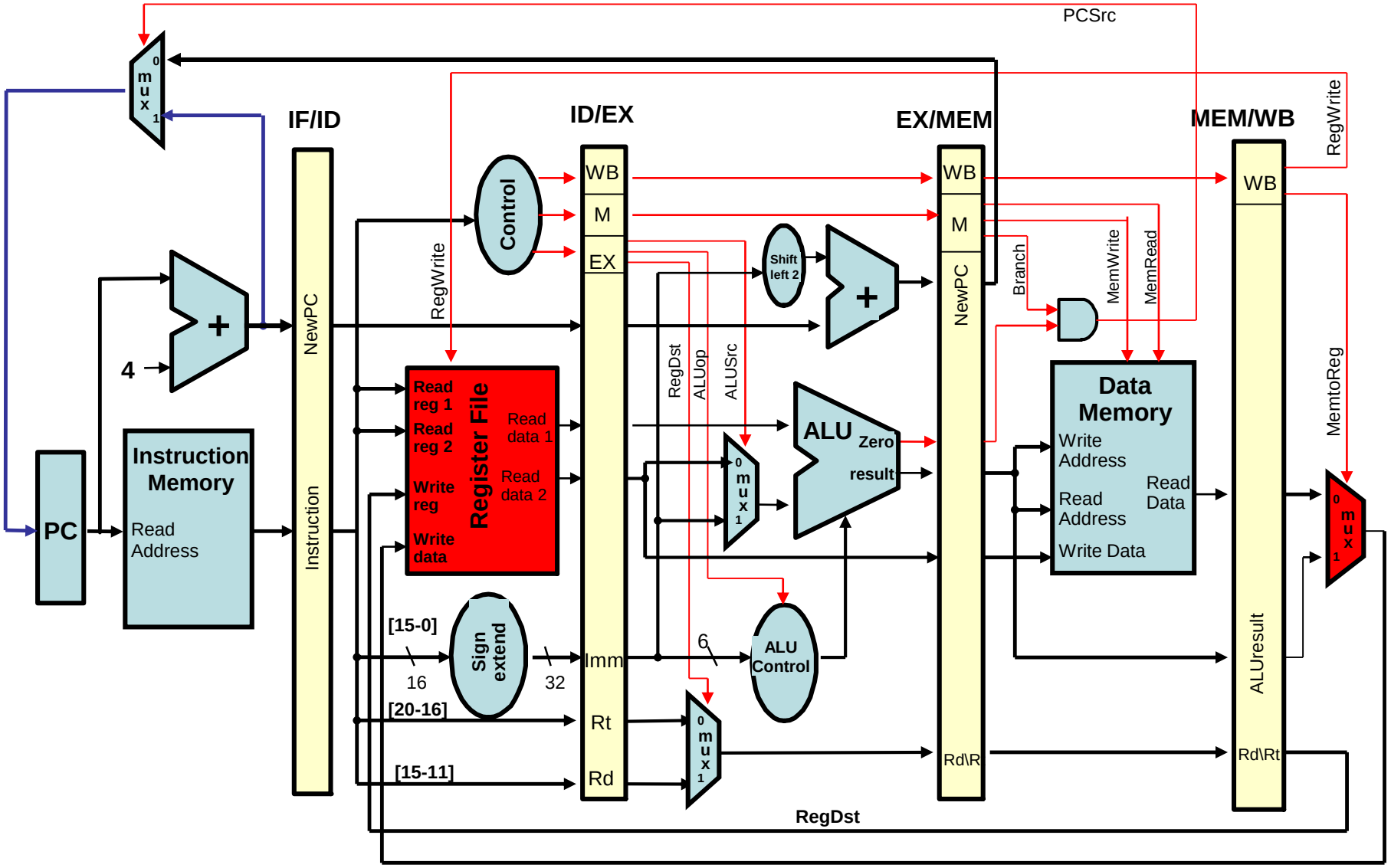
Execute


PCSrc
IF/ID
ID/EX
EX/MEM
MEM/WB
RegWrite
PC
Instruction Memory
Read Address
4
NewPC
Instruction
Control
WB
M
EX
Imm
Rt
Rd
Read reg 1
Read reg 2
Write reg
Write data
Register File
Read data 1
Read data 2
[15-0]
16
Sign extend
32
[20-16]
[15-11]
RegDst
ALUop
ALUSrc
Shift left 2
ALU
Zero
result
6
ALU Control
Branch
MemWrite
MemRead
Data Memory
Write Address
Read Address
Write Data
Read Data
ALUresult
Rd\R
Rd\Rt
MemtoReg
mux

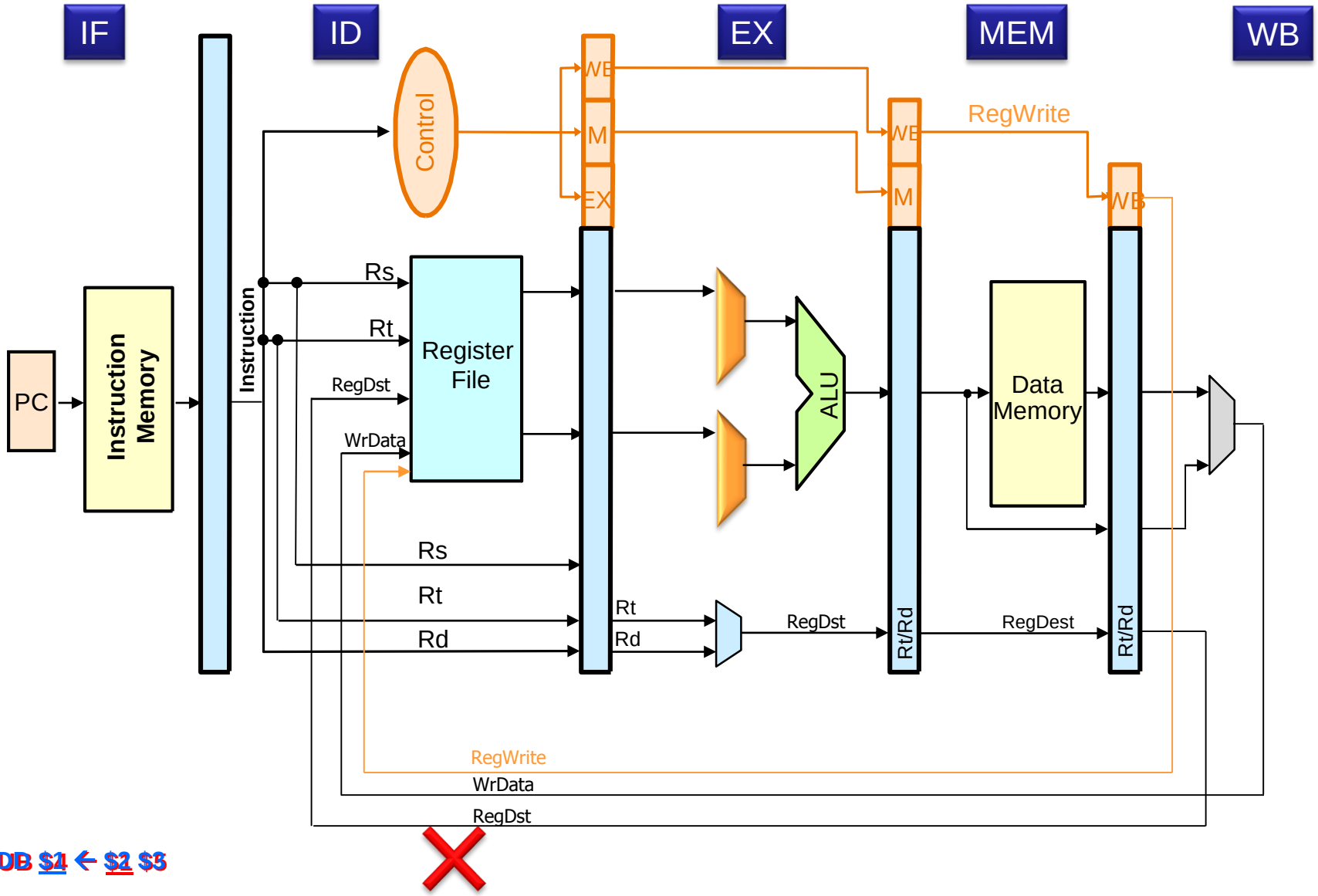
Memory


PCSrc
IF/ID
ID/EX
EX/MEM
MEM/WB
RegWrite
PC
Instruction Memory
Read Address
4
NewPC
Instruction
Control
WB
M
EX
Imm
Rt
Rd
Read reg 1
Read reg 2
Write reg
Write data
Register File
Read data 1
Read data 2
[15-0]
16
Sign extend
32
[20-16]
[15-11]
RegDst
ALUop
ALUSrc
Shift left 2
ALU
Zero
result
6
ALU Control
Branch
MemWrite
MemRead
Data Memory
Write Address
Read Address
Read Data
Write Data
MemtoReg
ALUresult
Rd\R
Rd\Rt

Writeback

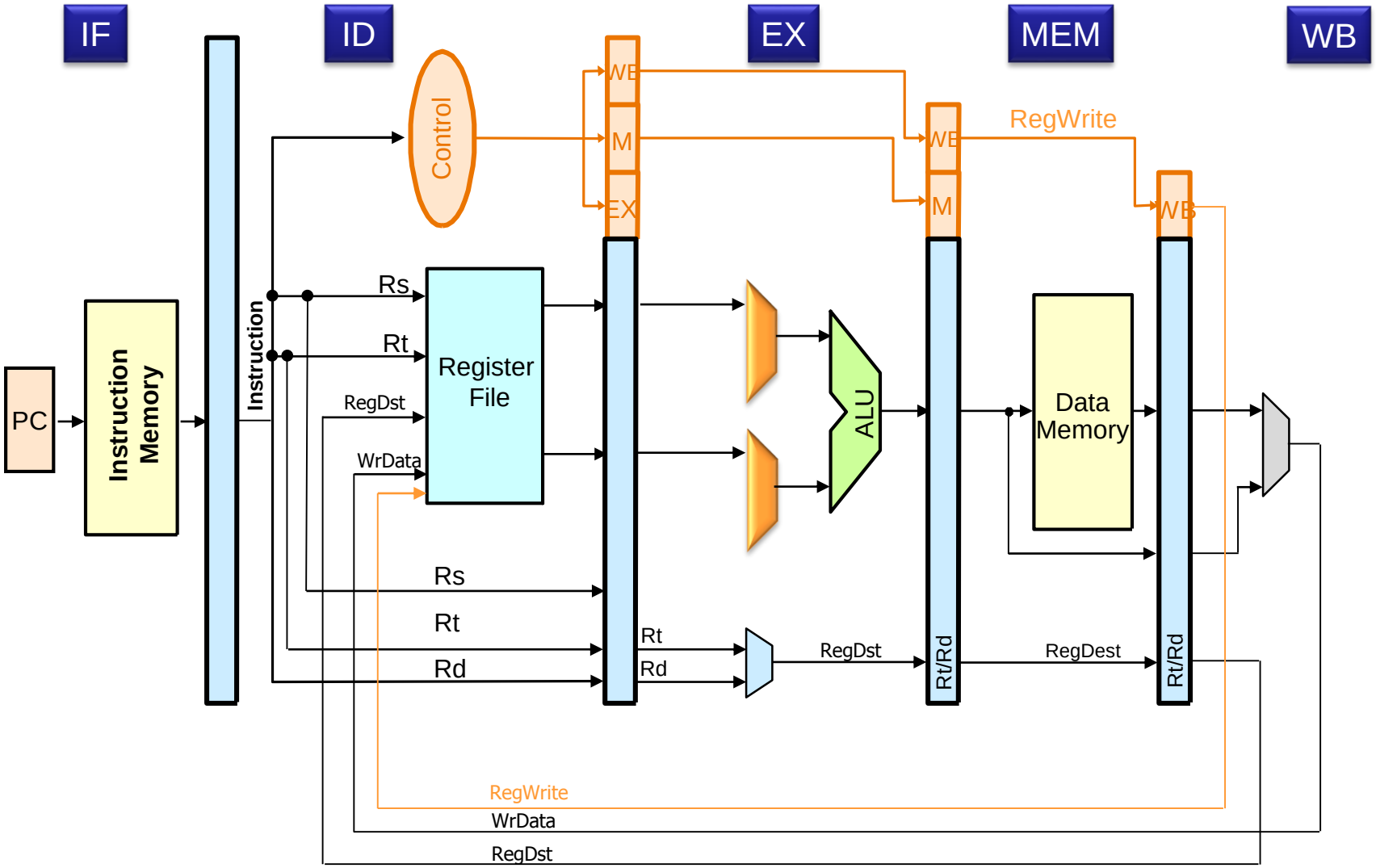

PCSrc
IF/ID
ID/EX
EX/MEM
MEM/WB
RegWrite
Control
WB
M
EX
Shift left 2
+
4
PC
Instruction Memory
Read Address
NewPC
Instruction
Read reg 1
Read reg 2
Write reg
Write data
Register File
Read data 1
Read data 2
RegDst
ALUop
ALUSrc
ALU
Zero
result
Branch
MemWrite
MemRead
Data Memory
Write Address
Read Address
Write Data
Read Data
MemtoReg
[15-0]
16
Sign extend
32
[20-16]
[15-11]
Imm
Rt
Rd
6
ALU Control
Rd\R
Rd\Rt
ALUresult
mux
0
1
RegDst

Data Hazard: RAW



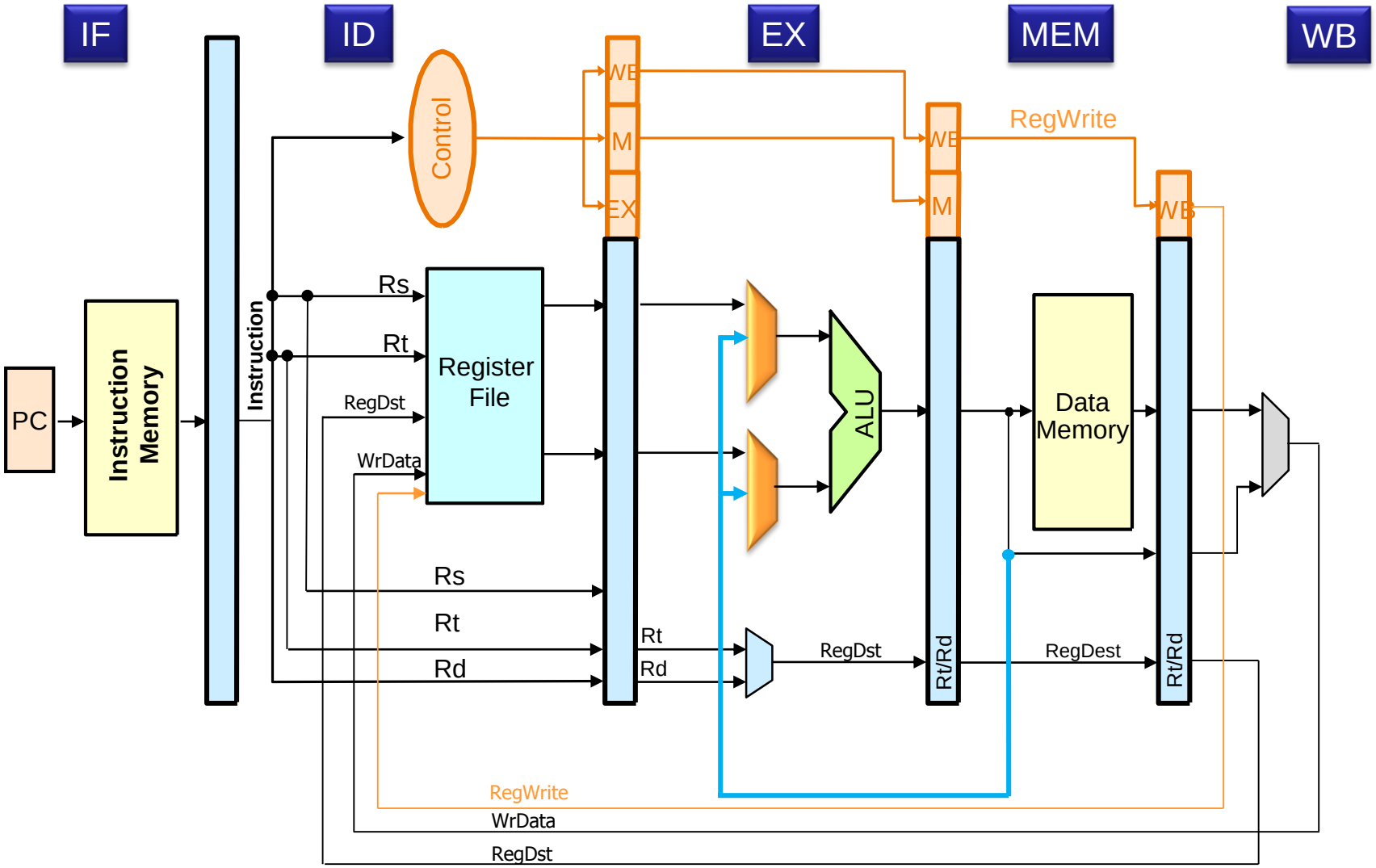
ADD $1 ← $2 $3

Data Hazard: Stall



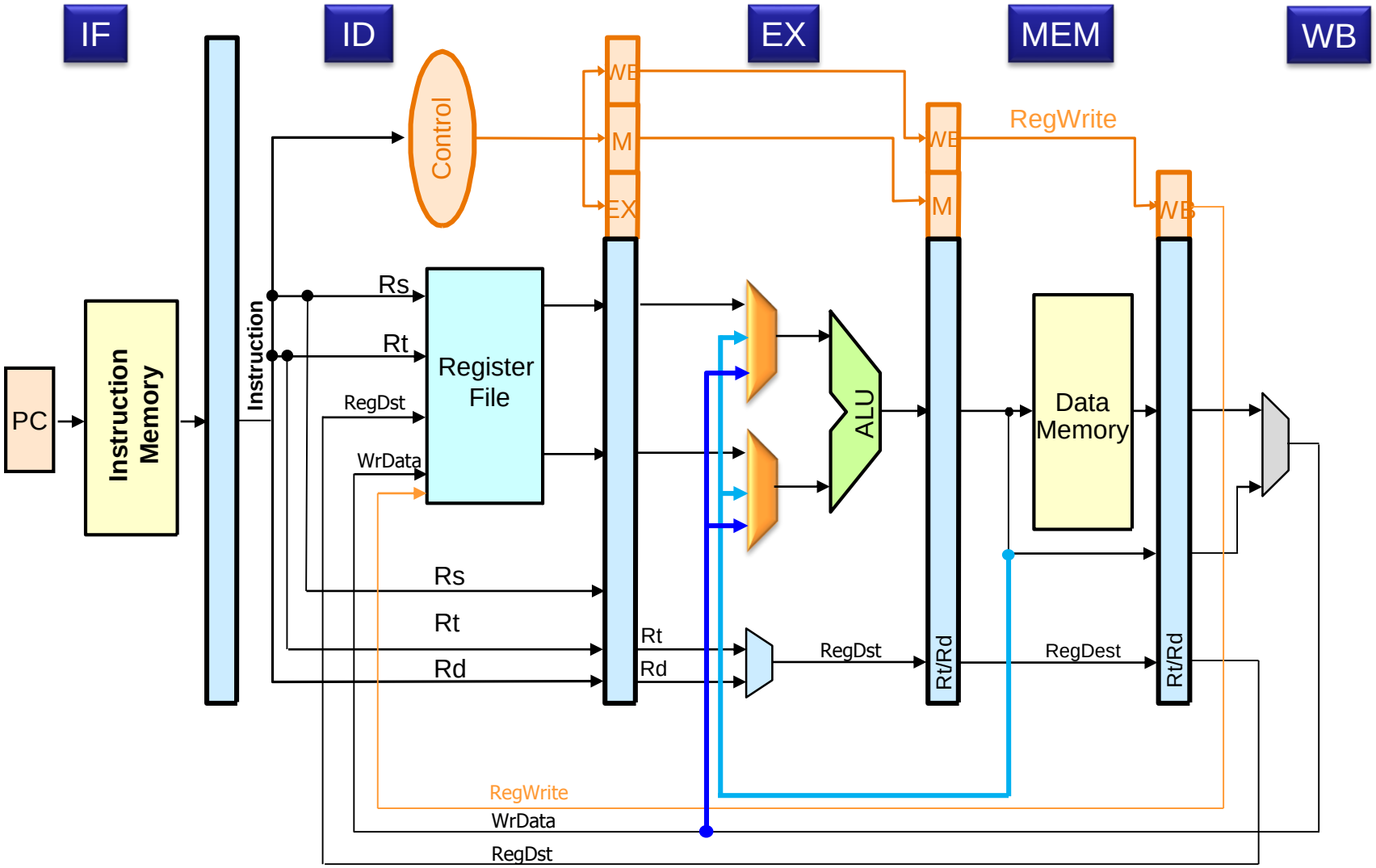
SUB $4 ← $1 $5 **ADD $1 ← $2 $3**

Data Hazard: Data Forwarding



SUB $4 ← $1 $5 **ADD $1 ← $2 $3**

Data Hazard: Data Forwarding

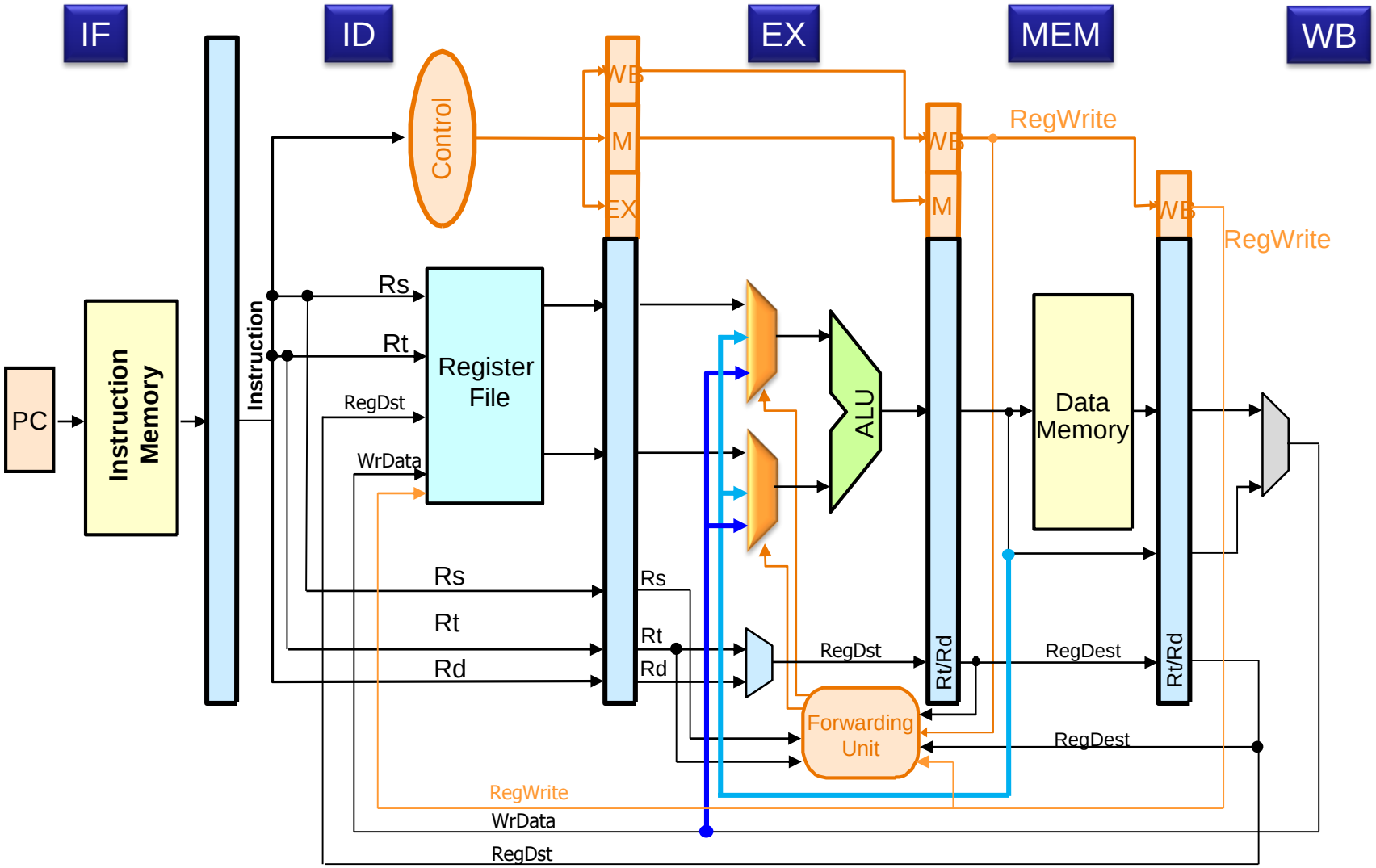


AND $8 ← $1 $5

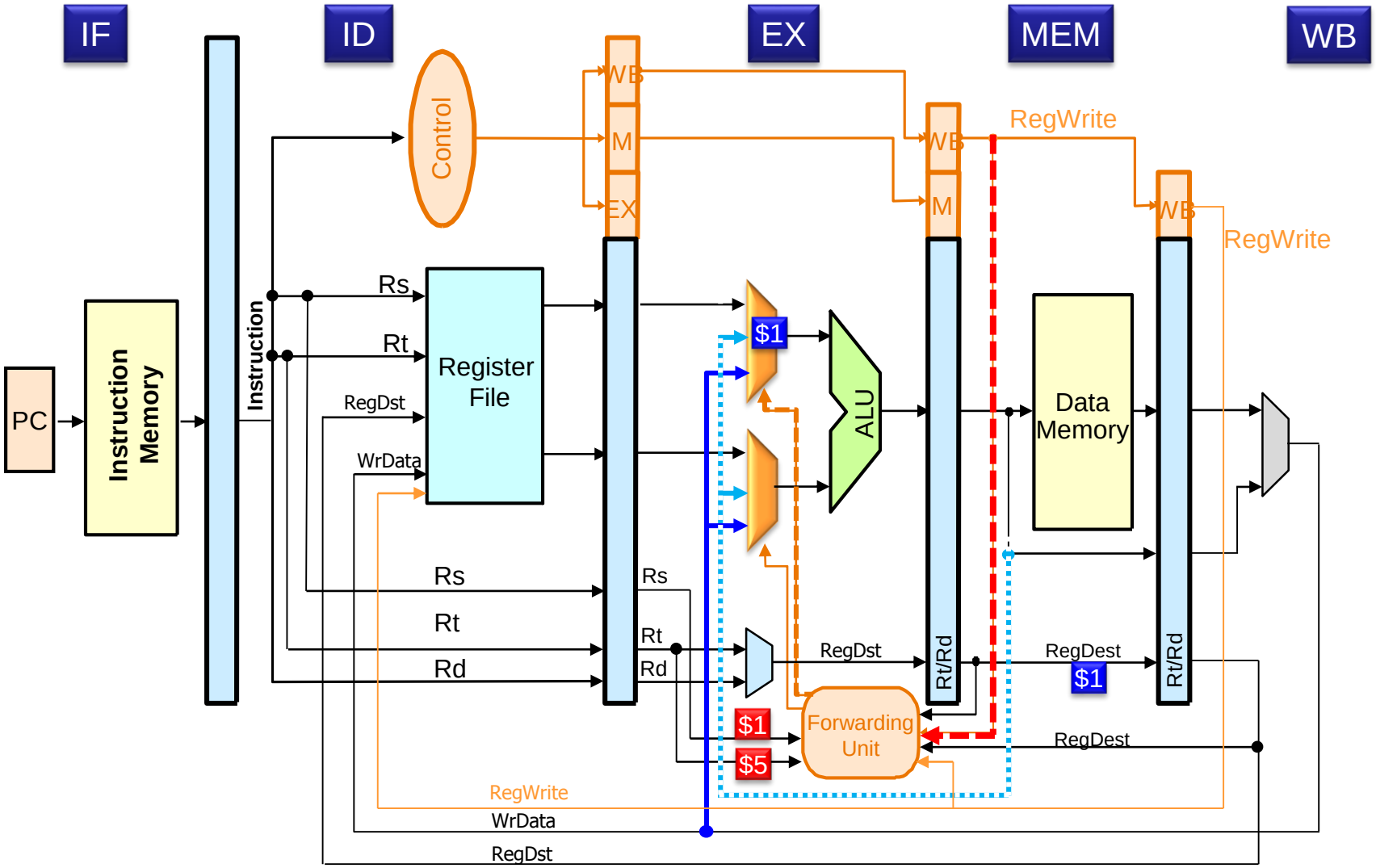
SUB $4 ← $2 $5

ADD $1 ← $2 $3

Forwarding Unit



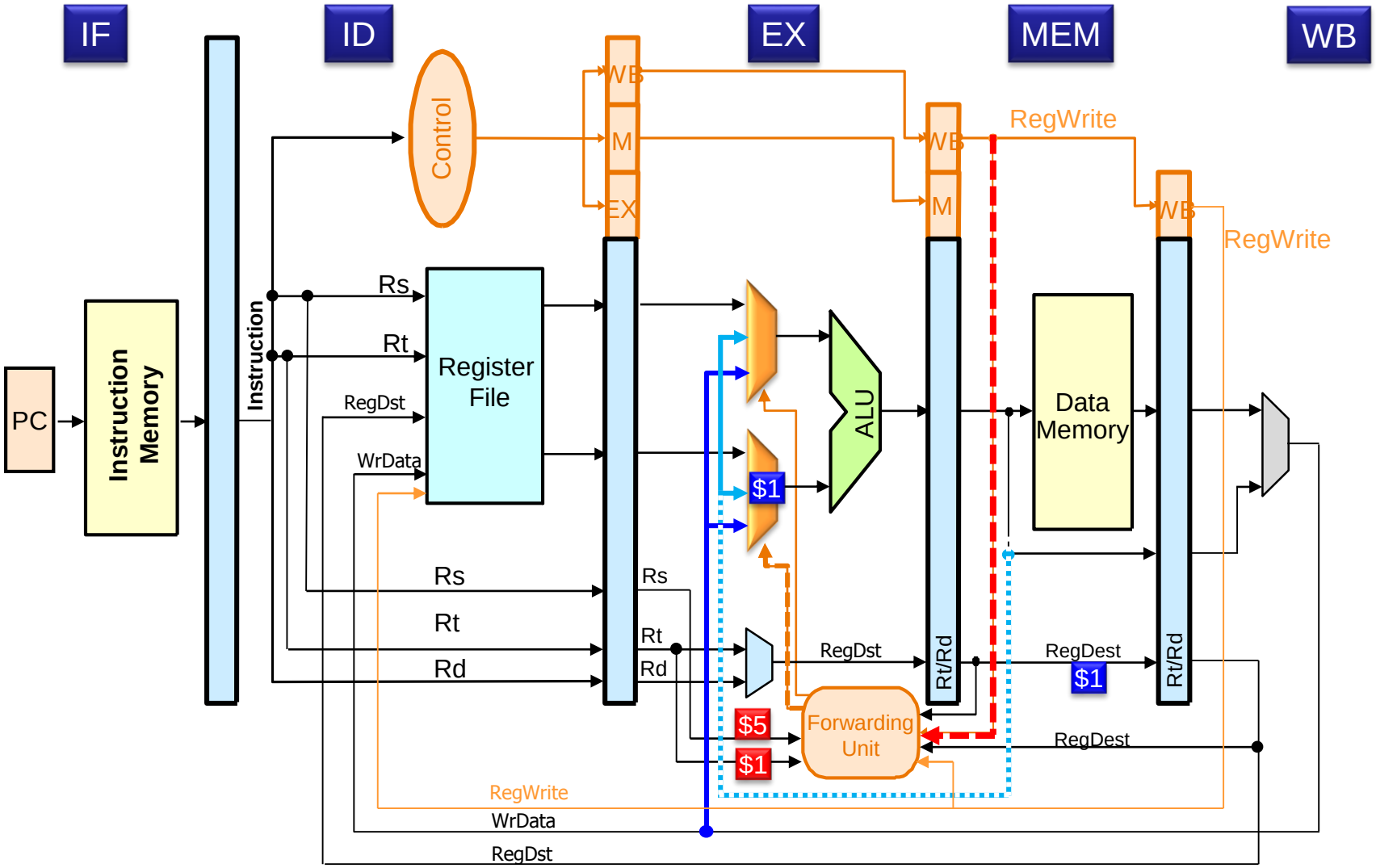
Forwarding Unit: example 1



SUB $4 ← $1 $5

ADD $1 ← $2 $3

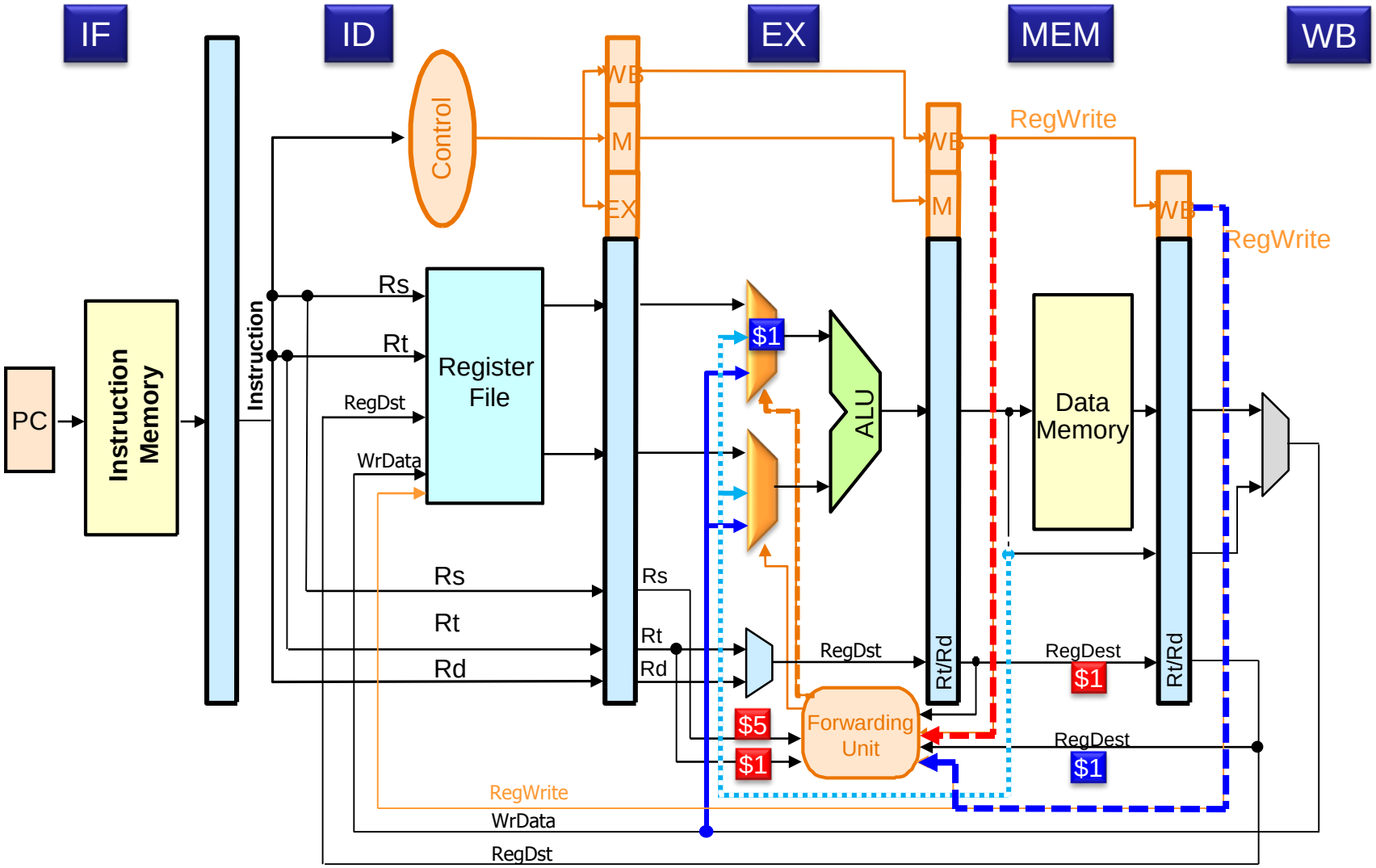
Forwarding Unit: example 2



SUB $4 ← $5 $1

ADD $1 ← $2 $3

Forwarding Unit: example 3

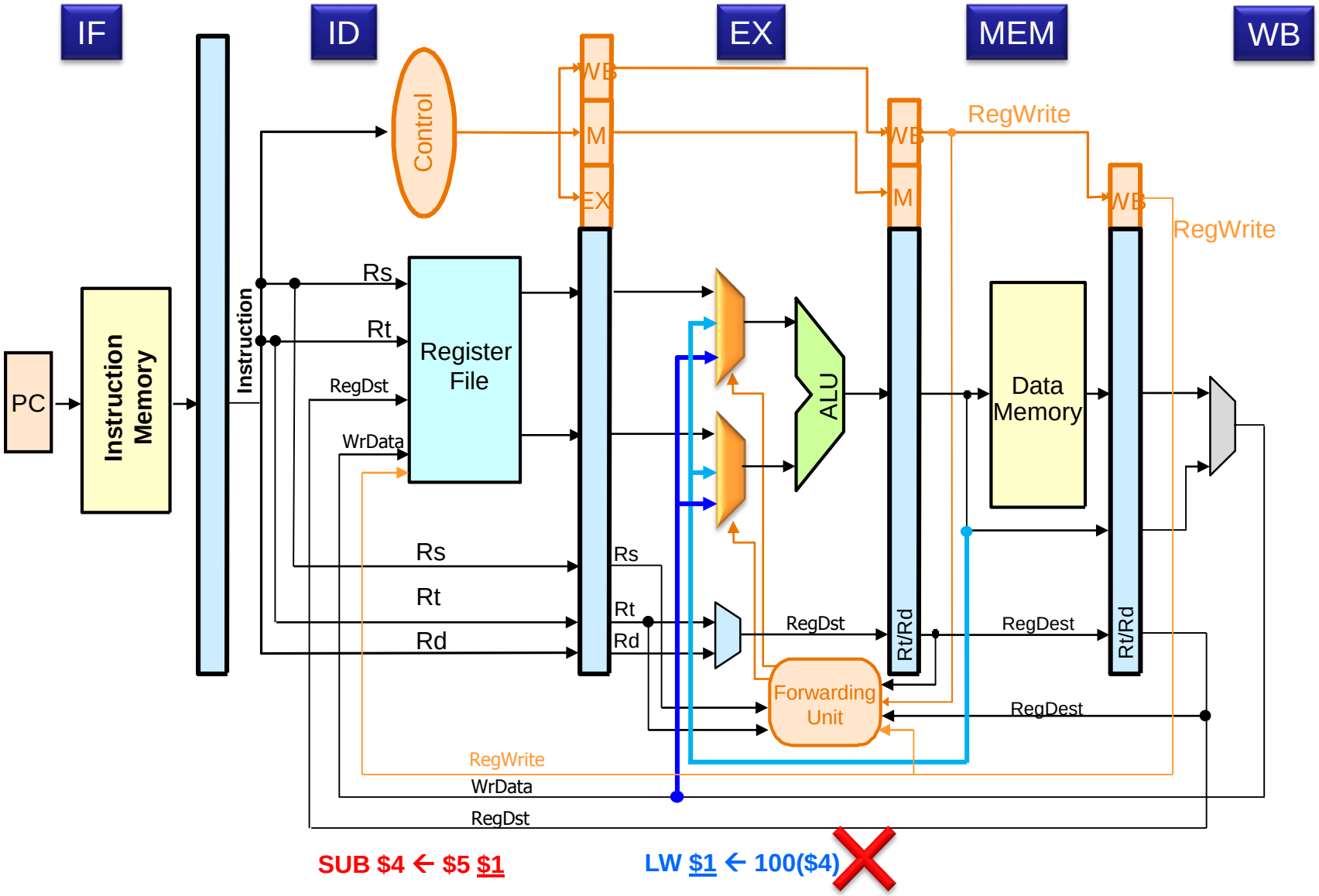


AND $8 ← $1 $5

SUB $1 ← $2 $5

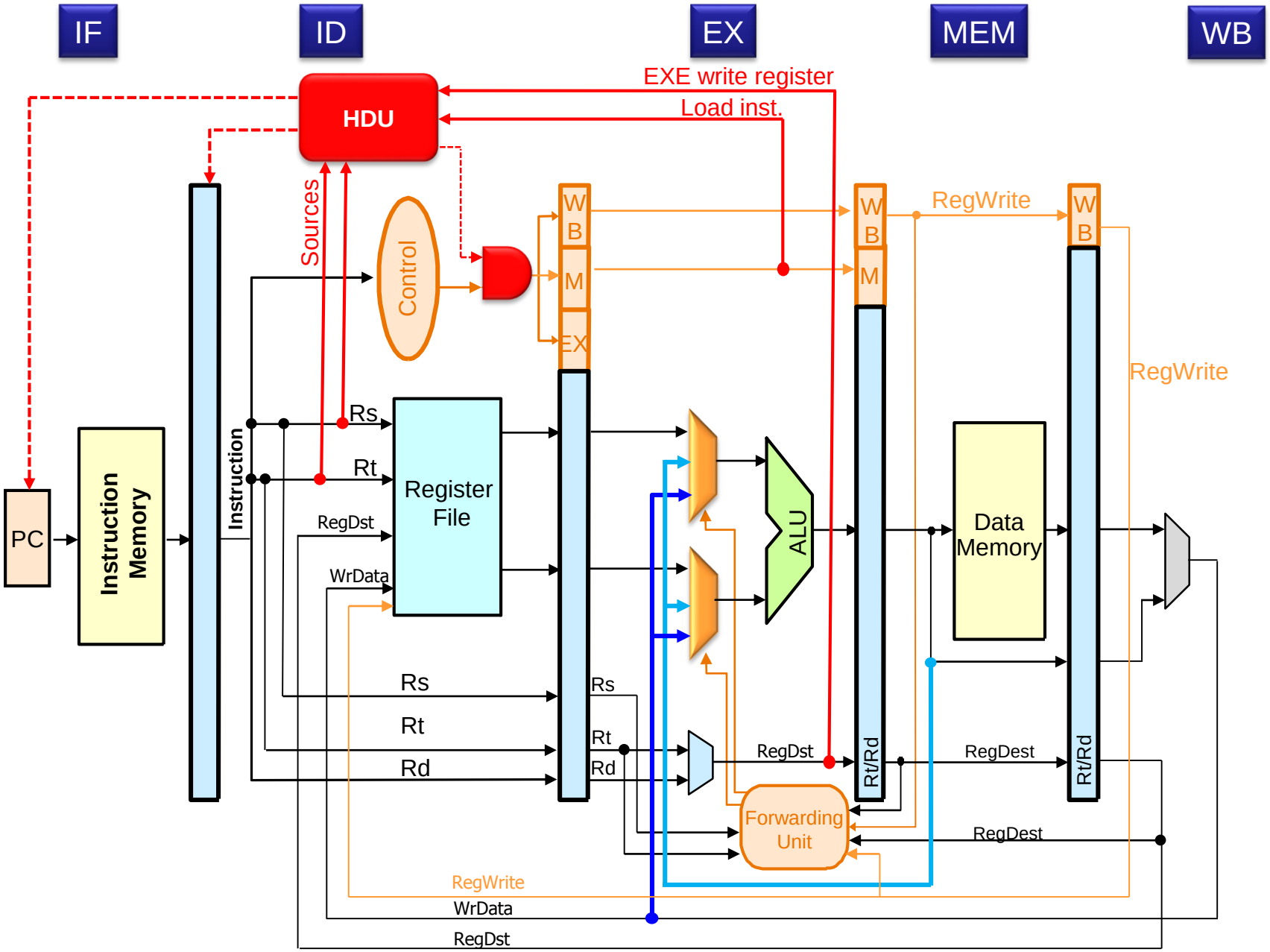
ADD $1 ← $2 $3

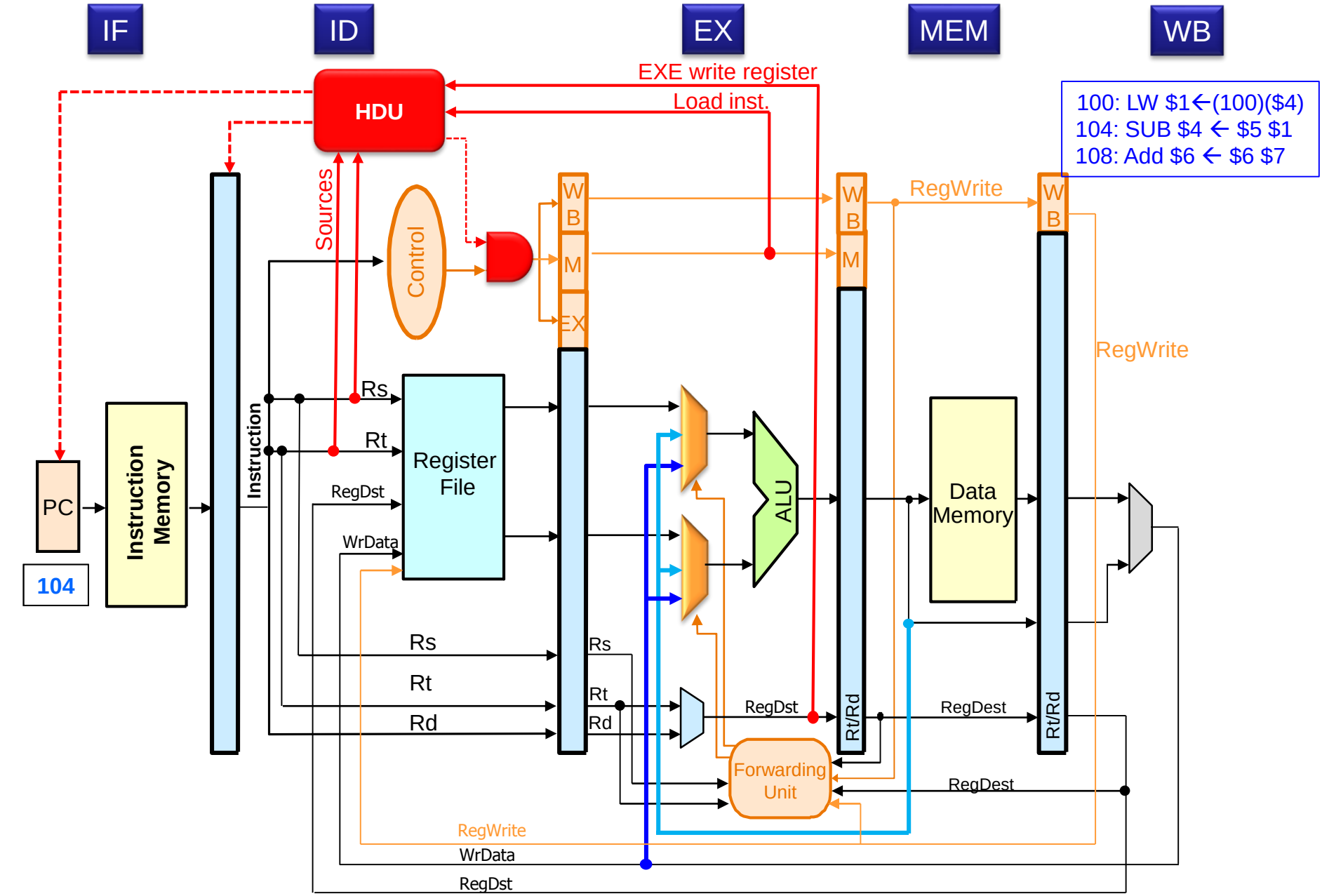
HDU: Hazard Detection Unit



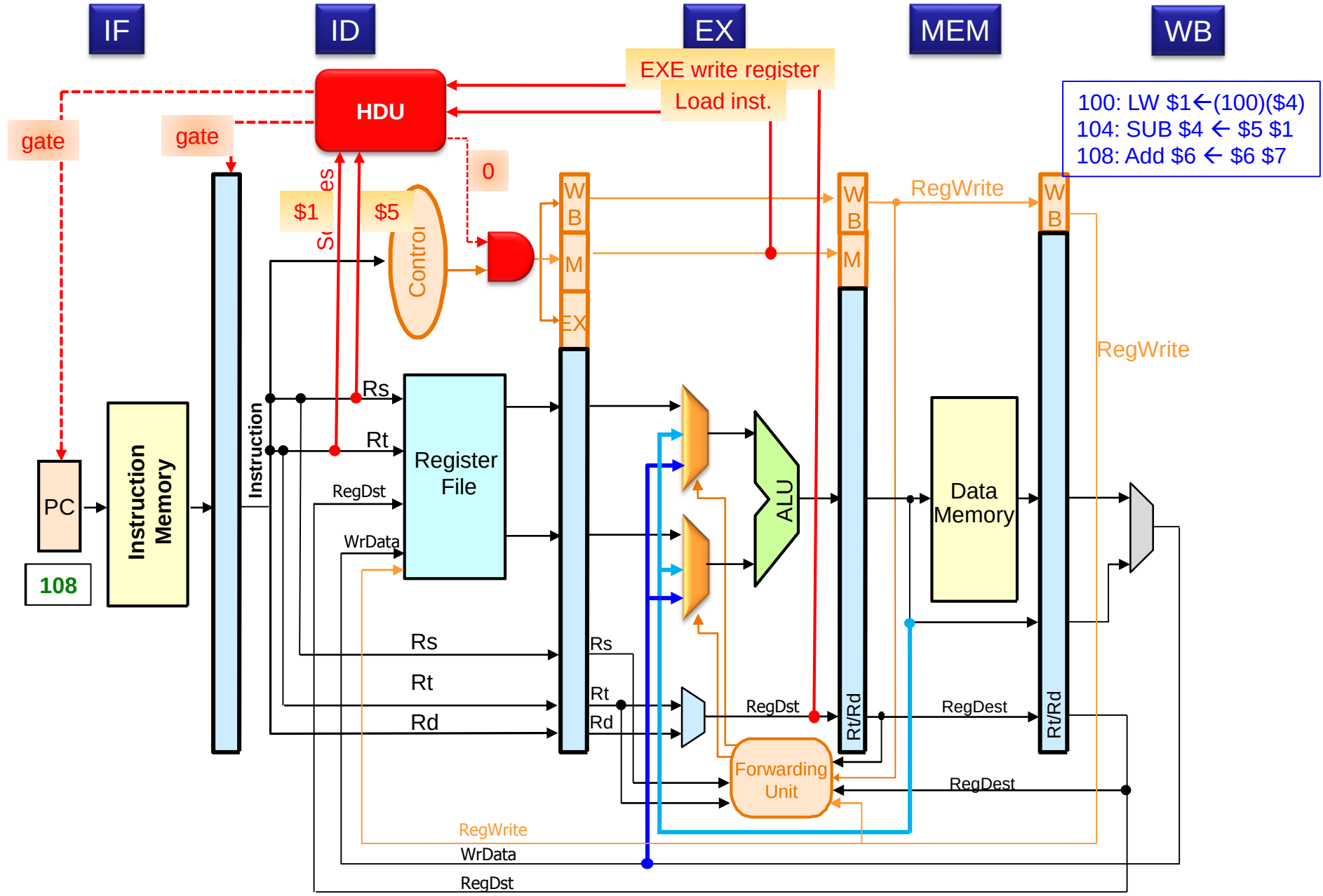
SUB $4 ← $5 $1

LW $1 ← 100($4)


IF
ID
EX
MEM
WB
HDU
EXE write register
Load inst.
Sources
Control
W
B
M
EX
RegWrite
RegWrite
PC
Instruction Memory
Instruction
Rs
Rt
RegDst
WrData
Register File
ALU
Data Memory
Rs
Rt
Rd
Rs
Rt
Rd
RegDst
Rt/Rd
RegDest
Forwarding Unit
RegDest
RegWrite
WrData
RegDst

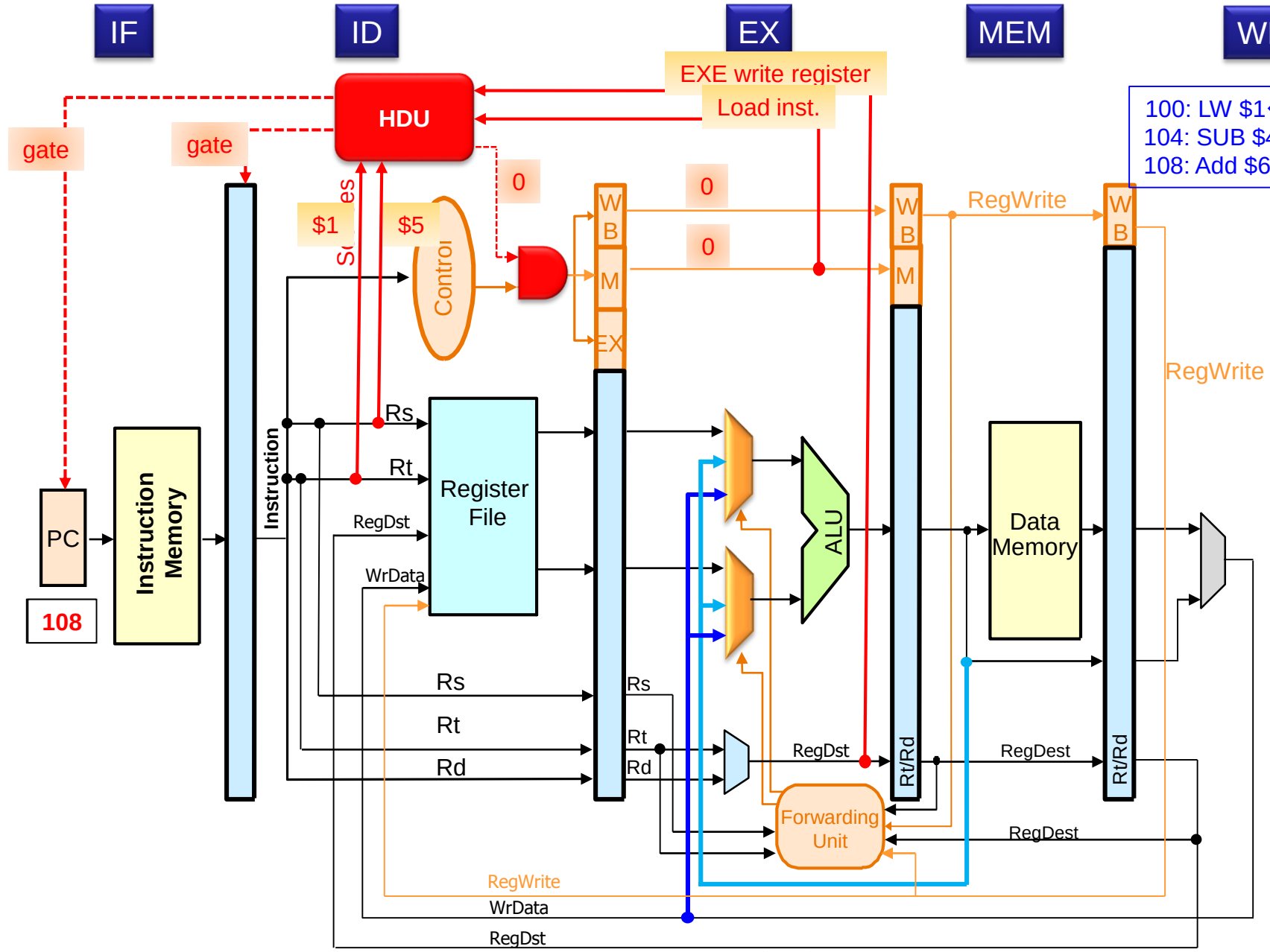

IF
ID
EX
MEM
WB
HDU
EXE write register
Load inst.
100: LW $1←(100)($4)
104: SUB $4 ← $5 $1
108: Add $6 ← $6 $7
Sources
Control
W
B
M
EX
RegWrite
RegWrite
PC
104
Instruction Memory
Instruction
Rs
Rt
RegDst
WrData
Register File
ALU
Data Memory
Rs
Rt
Rd
Rs
Rt
Rd
RegDst
Rt/Rd
RegDest
Rt/Rd
Forwarding Unit
RegDest
RegWrite
WrData
RegDst

SUB $4 ← $5 $1



ADD $6 ← $6 $7

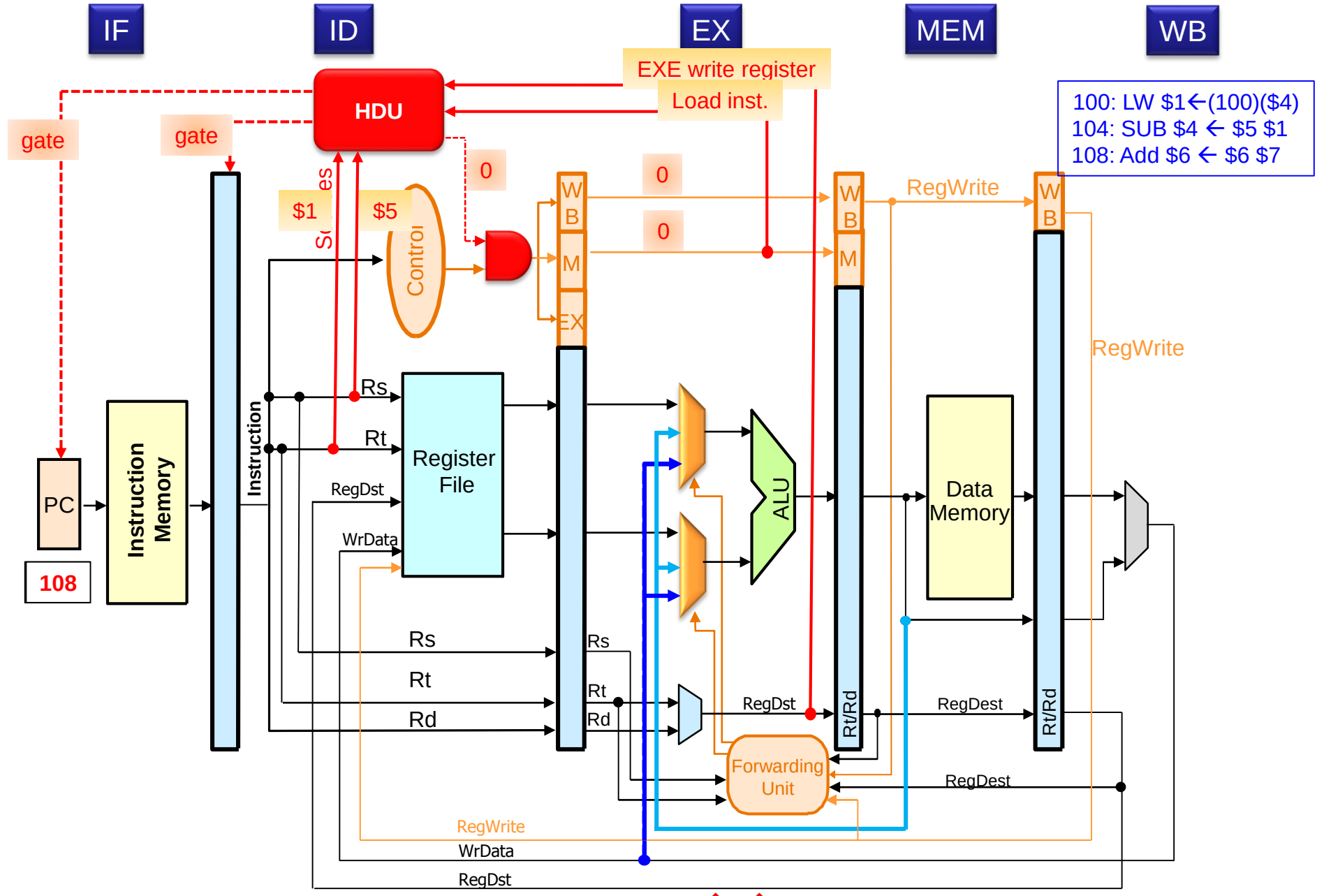
LW $1 ← 100($4)



ADD $6 ← $6 $7

SUB $4 ← $5 $1

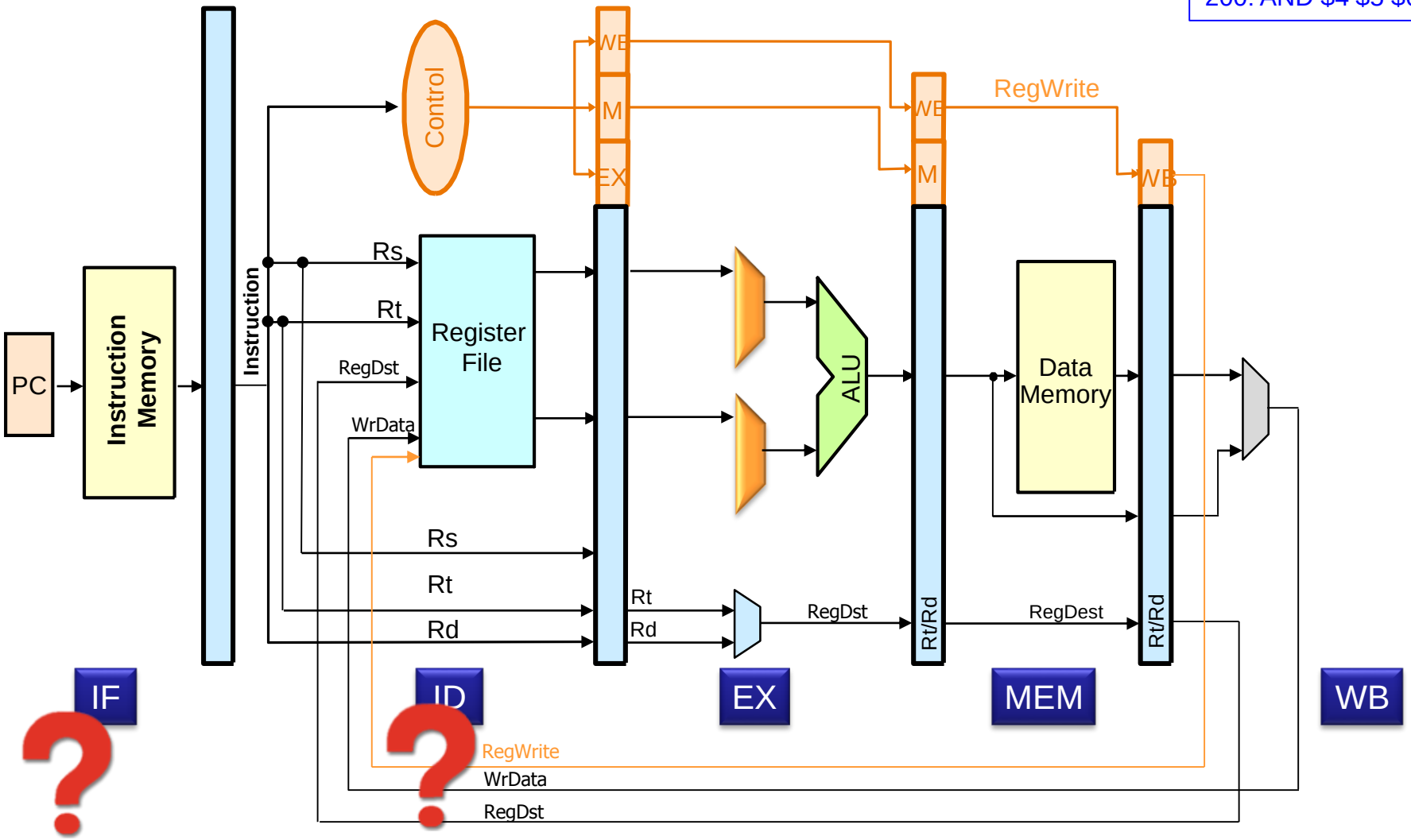
LW $1 ← 100($4)


IF
ID
EX
MEM
WB
EXE write register
Load inst.
HDU
gate
gate
100: LW $1←(100)($4)
104: SUB $4 ← $5 $1
108: Add $6 ← $6 $7
0
0
0
$1
$5
Control
WB
M
EX
RegWrite
RegWrite
PC
108
Instruction Memory
Instruction
Rs
Rt
RegDst
WrData
Register File
ALU
Data Memory
Rs
Rt
Rd
Rs
Rt
Rd
RegDst
RegDest
Rt/Rd
Forwarding Unit
RegDest
RegWrite
WrData
RegDst
ADD $6 ← $6 $7
SUB $4 ← $5 $1
SUB $4 ← $5 $1
LW $1 ← 100($4)

Control Hazard

Control Hazard

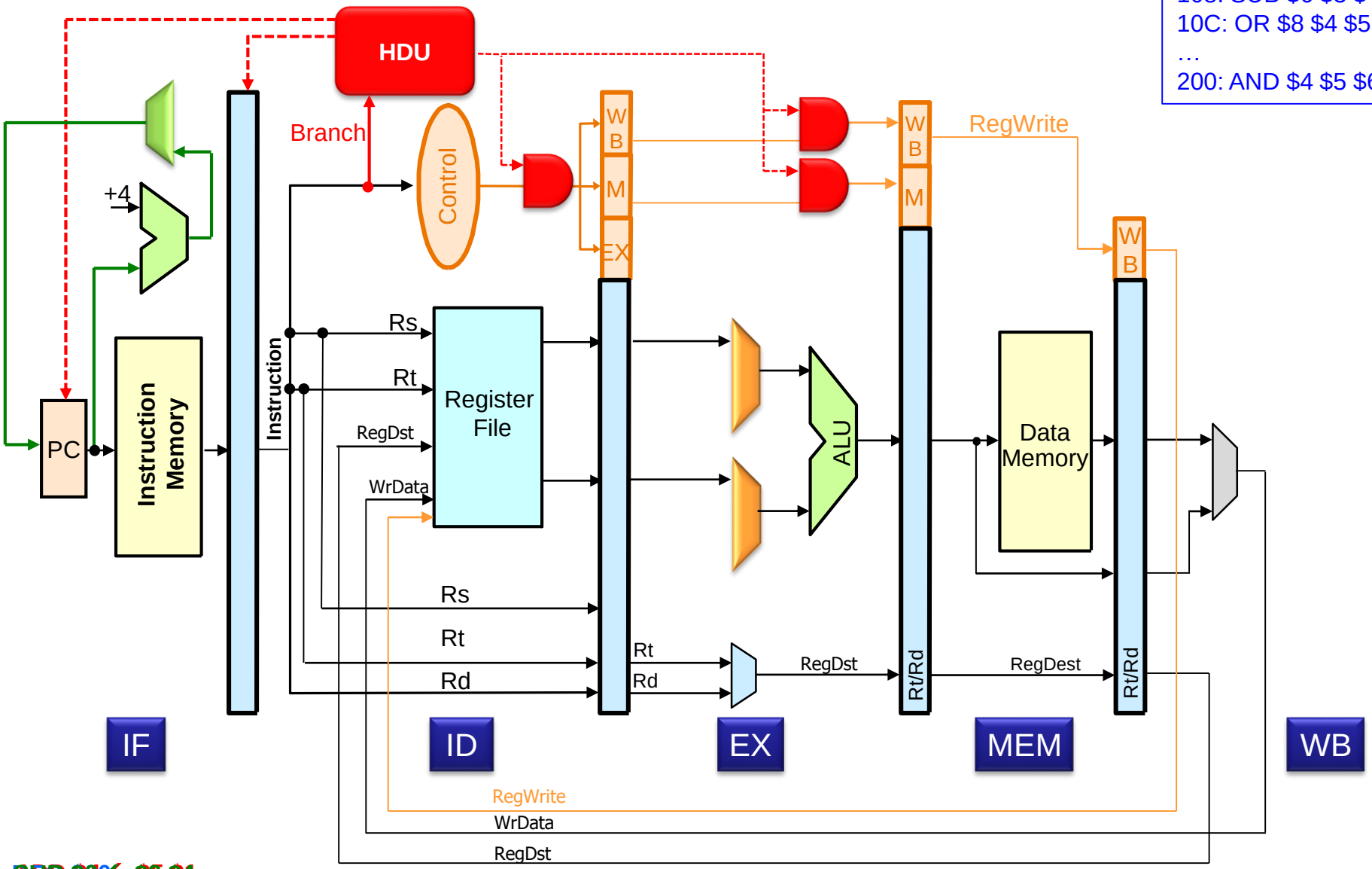
100: BEQ 200
104: ADD $4 $5 $1
108: SUB $6 $3 $4
10C: OR $8 $4 $5
…
200: AND $4 $5 $6



BEQ $6 ← $3 $4

Control Hazard: Stall…

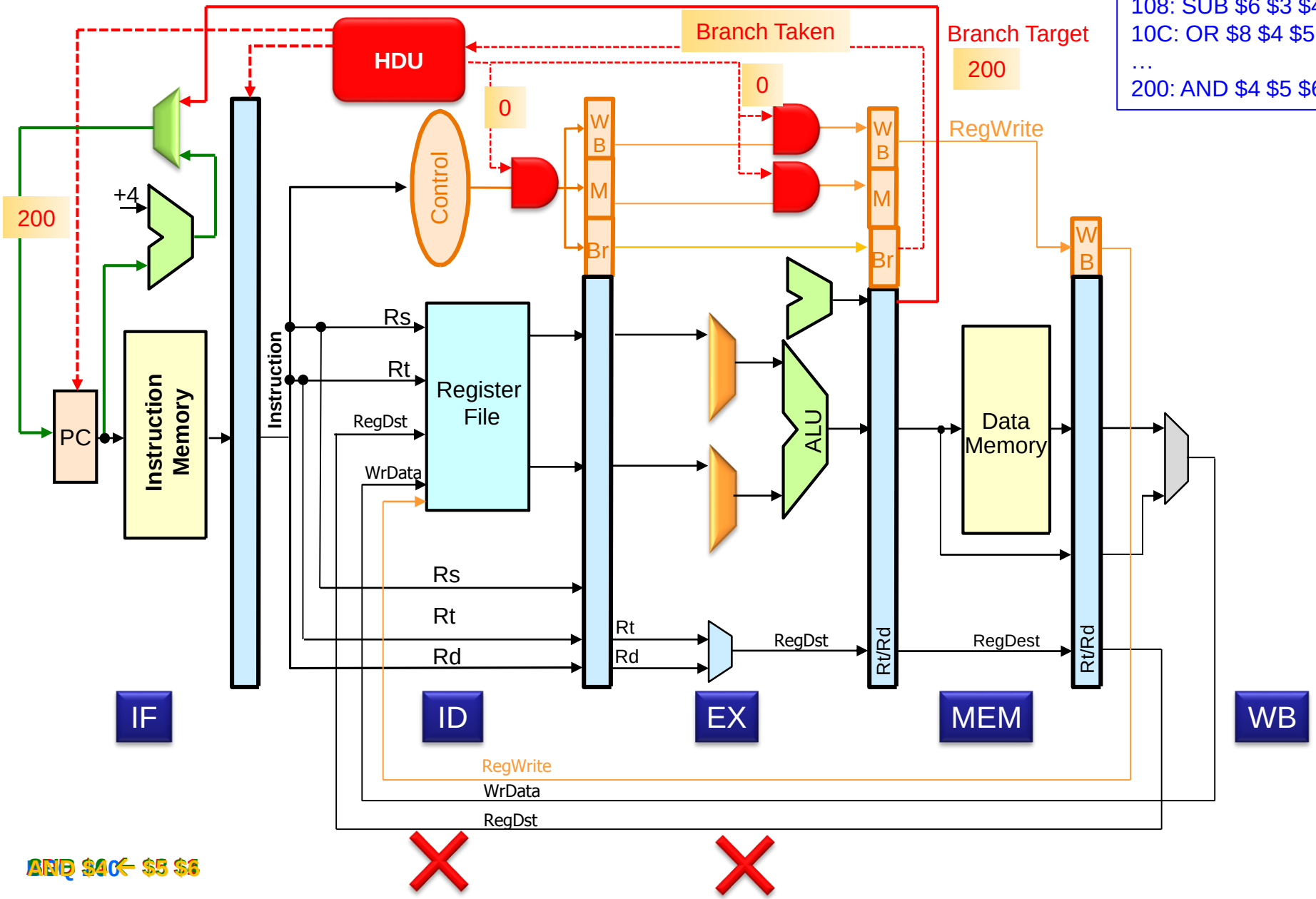
100: BEQ 200
104: ADD $4 $5 $1
108: SUB $6 $3 $4
10C: OR $8 $4 $5
…
200: AND $4 $5 $6



BEQ $6 ← $3 $4

Control Hazard: Assume Not Taken

100: BEQ 200
104: ADD $4 $5 $1
108: SUB $6 $3 $4
10C: OR $8 $4 $5
…
200: AND $4 $5 $6



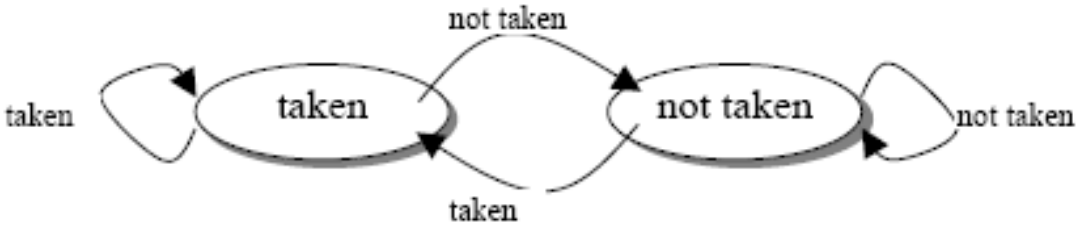
AND $4 ← $5 $6

פתרונות ל- Control Hazards (4)

שימוש במנגנון חיזוי BTB

Direction: (Taken/Not Taken)
מנגנון החיזוי "זוכר" את ההתנהגות של פקודת branch מסוימת בפעמים האחרונות.
אם הפקודה ביצעה את הקפיצות בפעמים האחרונות, נניח שגם כעת היא תבצע את הקפיצה.

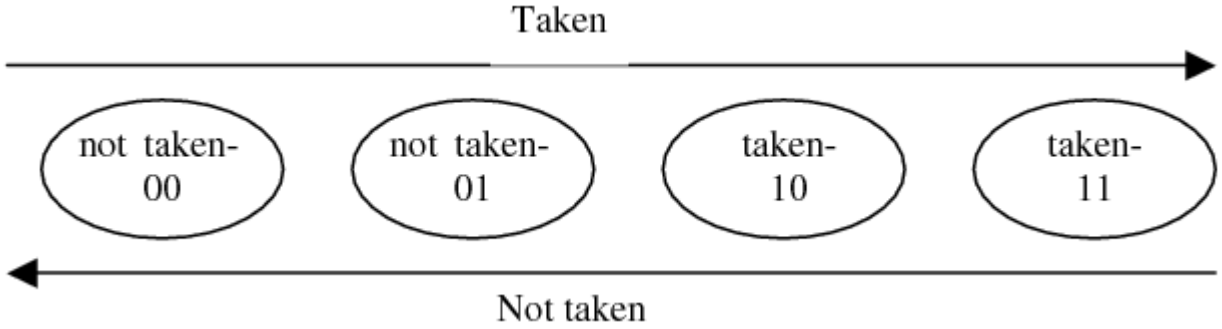
Target:
בנוסף להיסטוריה, מנגנון החיזוי "זוכר" גם את כתובת היעד של הקפיצה.
זכרון ההיסטוריה (עבור פקודת branch מסוימת) ממומש באמצעות מכונת מצבים.
לדוגמא:



כאשר צמותי הגרף מציינים את מצב החיזוי והקשתות את המעברים שיש לבצע עפ"י התוצאה (האמיתית) של פקודת ה-branch.
מצב התחלתי – not taken כי בפעם הראשונה בה נתקלים בפקודת branch לא ידועה כתובת היעד. רק לאחר שפקודת ה-branch תקפוץ בפעם הראשונה, נוכל לשמור עבורה את כתובת היעד.

פתרונות ל- Control Hazards (5)

מכונת המצבים הבאה, מספקת חיזוי אמין יותר:

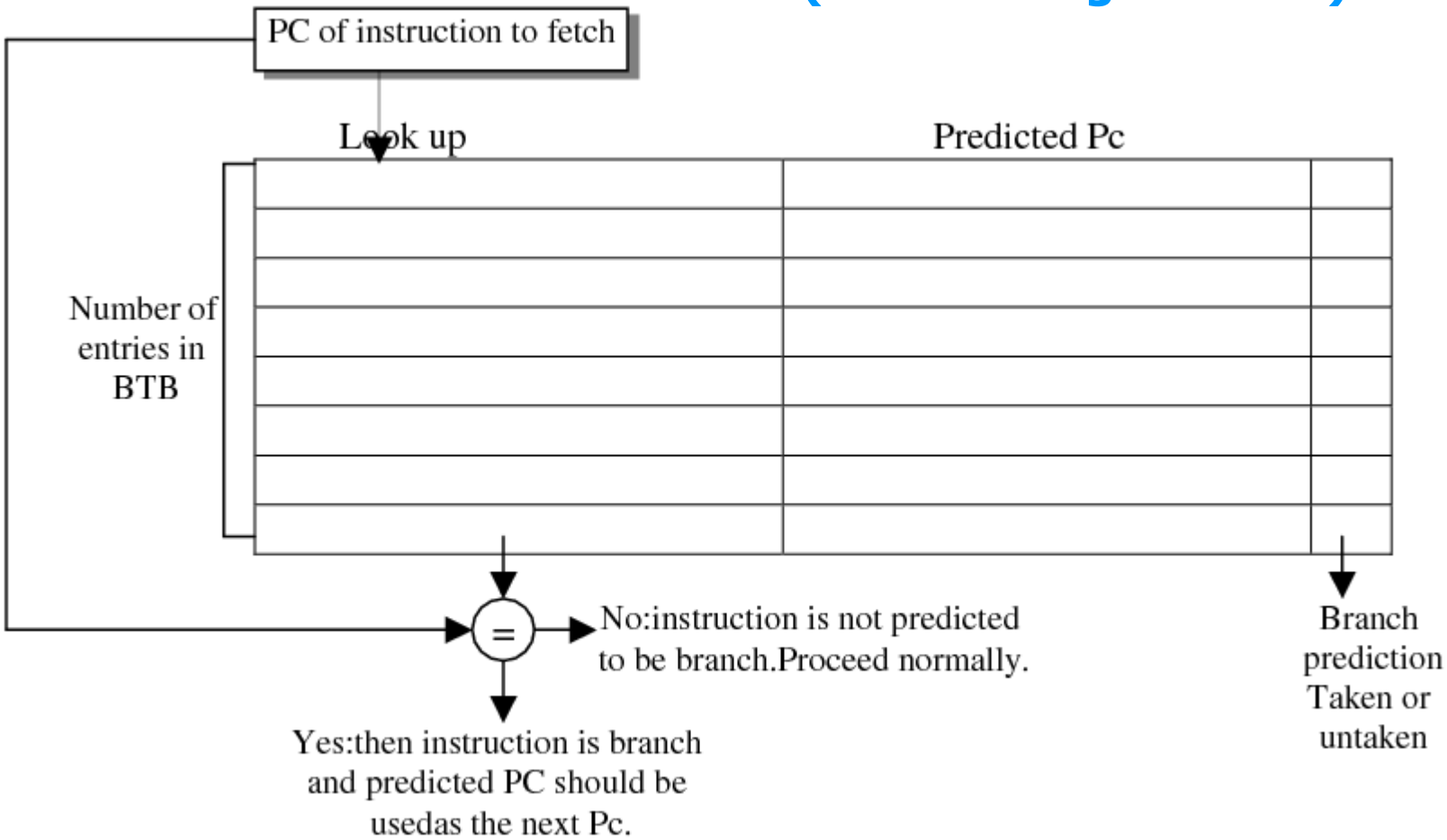


במכונה זו המצבים 00 ו-11 הנם מצבים יציבים.
המעבר מאחד ממצבים אלו למצב מנוגד לו יתרחש רק אם נשגה פעמיים בחיזוי.
בשגיאה הראשונה נעבור למצב זהה אך לא יציב, ורק בפעם השנייה שנשגה, נעבור למצב ההפוך.
שימוש במצבים כאלו מונע טעויות בחיזוי הנובעות מקפיצות / אי-קפיצות חריגות.

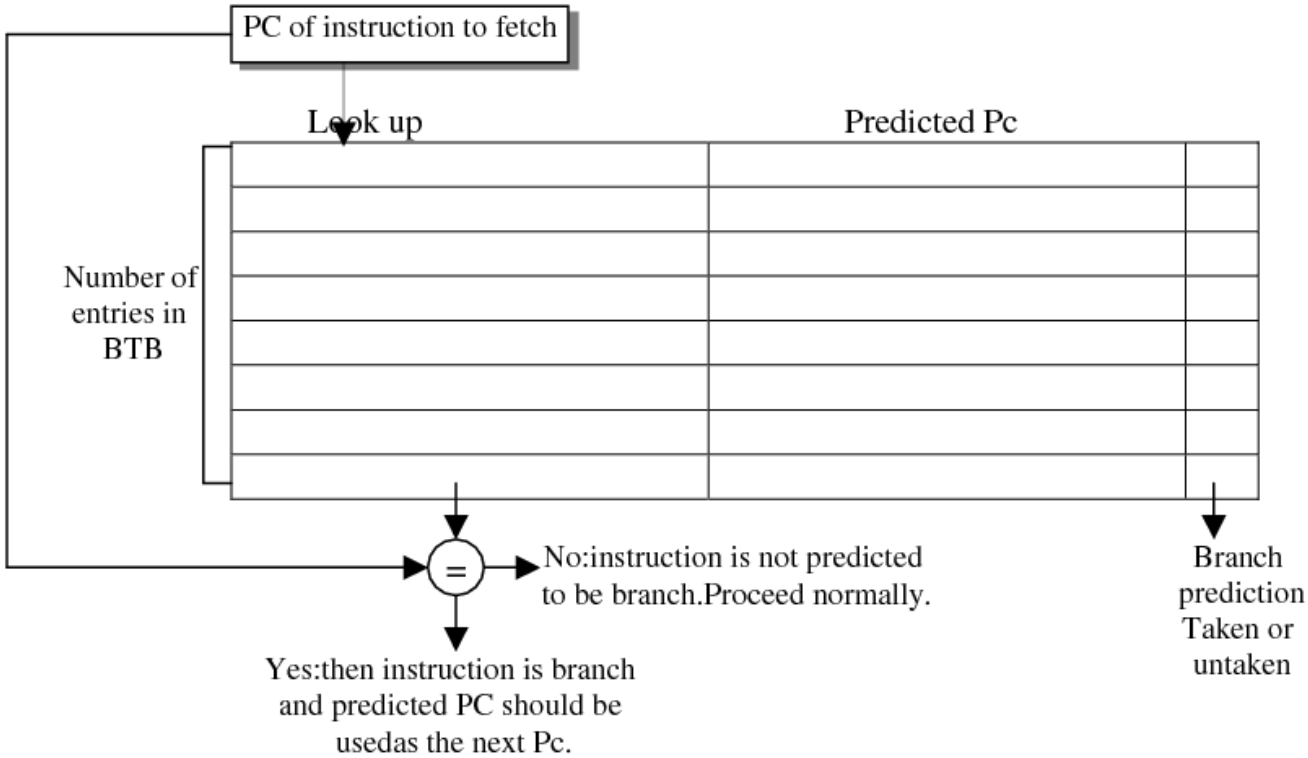
למשל, בסיום ביצוע לולאה נחזה שתתרחש קפיצה, אך נשגה ונעבור למצב taken לא יציב. בפעם הבאה בה נחזור לבצע את הלולאה שוב, נחזה שתתרחש קפיצה ואכן בד"כ חיזוי זה יהיה נכון.
במכונה הקודמת (עם שני מצבים) לאחר השגיאה הראשונה נעבור למצב not taken, אזי בפעם הבאה שנגיע ללולאה נחזה not taken, וסביר להניח שבזאת נשגה שוב.
אי-קפיצה חריגה גרמה לנו לבצע שני חיזויים שגויים, לעומת חיזוי שגוי אחד במכונה עם המצבים היציבים.
בתור המצב ההתחלתי באוטומט כזה נקבע 01 – not taken לא יציב.

(6) Control Hazards -פתרונות ל

מבנה BTB (Branch Target Buffer)



פתרונות ל-
Control Hazards (7)



מבנה BTB (Branch Target Buffer)

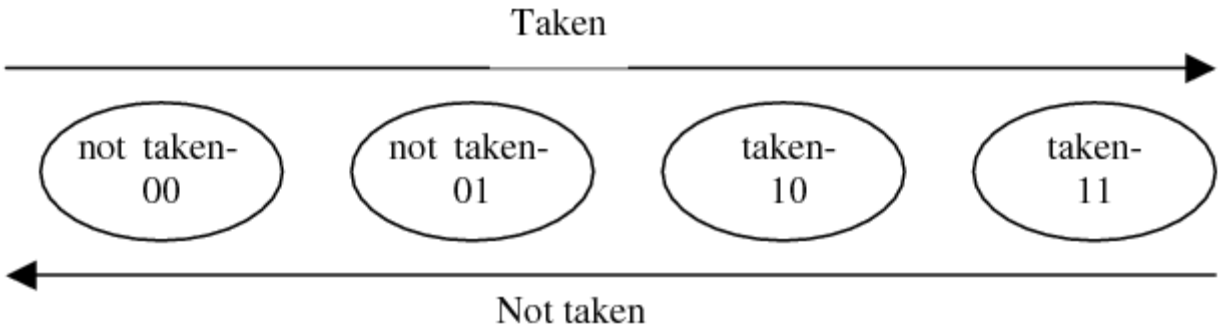
בשדה השמאלי נשמרת כתובת פקודת ה-branch שמזהה את הפקודה.

לפעמים לא מחזיקים את הכתובת המלאה, אלא משתמשים ב-hash table.
שימוש ב-hash table יכול ליצור מצב בו מספר פקודות branch ממופות לאותה הכניסה בטבלה וכתוצאה מכך ייתכן החיזוי שגוי.

בשדה האמצעי נשמרת כתובת היעד. כאמור, כתובת זו יכולה להתווסף לטבלה רק לאחר שפקודת ה-branch בוצעה לפחות פעם אחת.

דוגמא

נתון מעבד בעל חמישה שלבי pipeline IF,ID,EX,MEM,WB עם מנגנון חיזוי (BTB) הפועל ע"פ האלגוריתם הבא:



כאשר נרשמת פקודת branch ב-BTB בפעם הראשונה, מצב החיזוי שלו מאותחל ל-01.

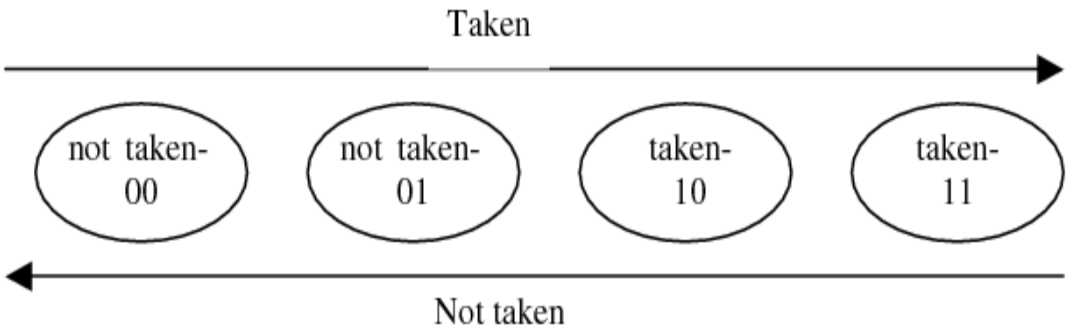
מה יהיה החיזוי בכל פעם שפקודת ה BNEQ שבקטע הבא מבוצעת:

```
        MOVI R1,2
loop1:  MOV R2,R1
loop2:  Dec R2
        BNEQ R2,R0,loop2
        INC R1
        BLT R1,4,loop1
```

מהי כמות השגיאות בחיזוי?

דוגמא (2)

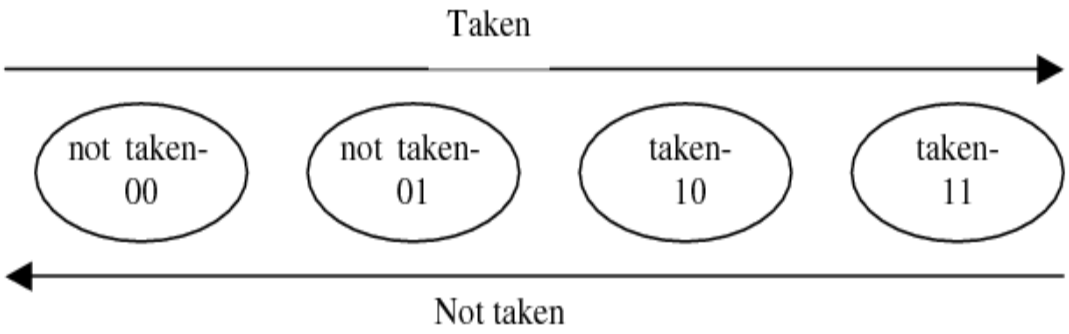
```
        MOVI R1,2
loop1:  MOV R2,R1
loop2:  Dec R2
        BNEQ R2,R0,loop2
        INC R1
        BLT R1,4,loop1
```



	מחזור	R1	R2	מצב נוכחי	חיזוי	בפועל	המצב הבא	שגיאה?

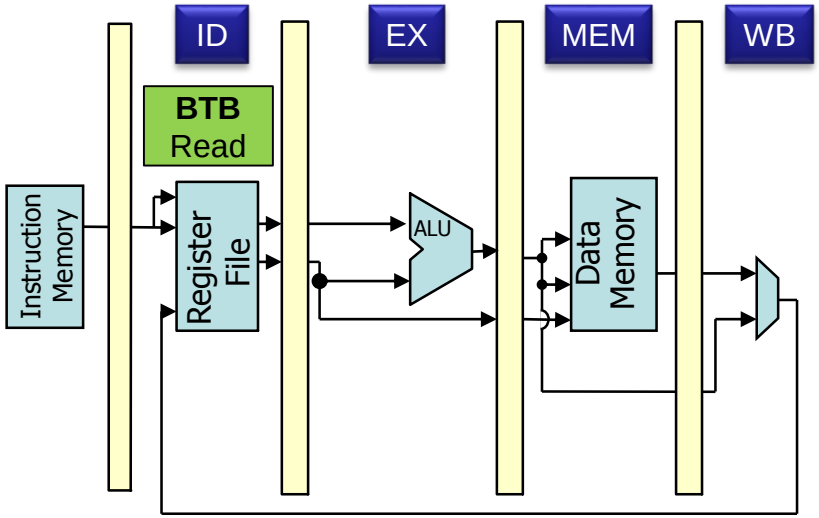
דוגמא (2)

```
        MOVI R1,2
loop1:  MOV R2,R1
loop2:  Dec R2
        BNEQ R2,R0,loop2
        INC R1
        BLT R1,4,loop1
```



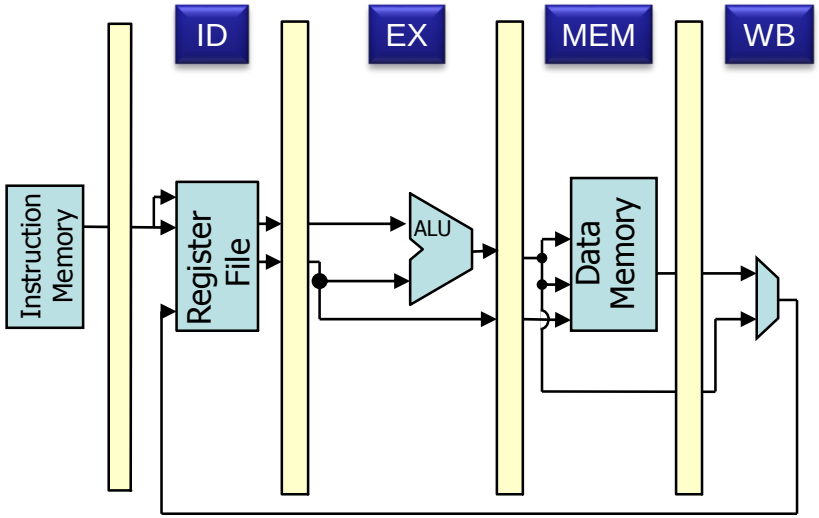
	מחזור	R1	R2	מצב נוכחי	חיזוי	בפועל	המצב הבא	שגיאה?
Loop1	1	2	2					
BNEQ	2	2	1	01	not taken	taken	10	+
BNEQ	3	2	0	10	taken	not taken	01	+
Loop1	4	3	3					
BNEQ	5	3	2	01	not taken	taken	10	+
BNEQ	6	3	1	10	taken	taken	11	
BNEQ	7	3	0	11	taken	not taken	10	+

דוגמא (3)



נניח שהחיפוש ב BTB מבוצע בשלב ה- **ID** (של ה- branch) ובאותו שלב נטען ה- PC הנכון.
במקרה של BTB hit, כאשר החיזוי הוא Taken, הפקודה מכתובת היעד מובאת במחזור שאחר-כך (שלב EX שלו).
הכרעת הקפיצה מבוצעת בשלב ה- EXE ובמקרה של חיזוי שגוי מבוצע pipeline flush.

דוגמא (3)



ב. כמה מחזורי שעון מבוזבזים במקרה של חיזוי נכון, וכמה בחיזוי שגוי?

עבור חיזוי נכון , כאשר ה- branch הוא NT : אנו טוענים באופן רגיל פקודות באופן סדרתי כרגיל ולכן איננו מבזבזים מחזורי שעון.

עבור חיזוי נכון , כאשר ה- branch הוא T : החיפוש ב BTB וטעינת ה- PC הנכון נעשה בשלב ה- ID במקום בשלב ה- IF.
לכן לא נוכל לבצע fetch לפקודה הבאה שאמורה להתבצע ללא השהיית ה - pipeline למשך מחזור שעון אחד.
לכן נבזבז מחזור שעון יחיד.

עבור חיזוי שגוי:הכרעת הקפיצה מבוצעת בשלב ה EX במקום בשלב ה- MEM. לכן נפסיד 2 מ"ש על ביצוע ה- flush .

דוגמא (4)

בחיזוי נכון:

beq:	IF	ID	EX	MEM	WB		
dest:		stall	IF	ID	EX	MEM	WB

נבצע stall יחיד שאינו תלוי בנכונות החיזוי, אלא נובע מהעובדה שהחיפוש ב BTB נעשה בשלב ה- ID. במעבד המקורי לא היה צורך בהשהיית ה- pipeline.

בחיזוי שגוי:

beq:	IF	ID	EX	MEM	WB			
שגוי:		stall	IF	ID	EX	MEM	WB	
dest:				IF	ID	EX	MEM	WB

נפסיד שני מחזורי שעון בלבד מכיוון שהכרעת הקפיצה מבוצעת בשלב EX

במעבד ה MIPS המקורי השהנו את ה- pipeline למשך שלושה מחזורי שעון כי הכרעת הקפיצה בוצעה בשלב ה MEM.

דוגמא (4)

ג. הצע דרכים שיאפשרו להימנע מה- stall במקרה של חיזוי נכון.

פתרון מספר 1:
פקודות ה branch מזוהות ב BTB ע"י הכתובת שלהן. אם במקום כתובת פקודת ה-branch נשמור ב- BTB את כתובת הפקודה שמתבצעת לפני פקודת ה branch נוכל להימנע מביצוע stall במקרה של חיזוי נכון, כי כשהפקודה שלפני פקודת branch תגיע לשלב ה ID יחל החיפוש ב- BTB בעוד שפקודת ה branch נמצאת בשלב ה IF. זהו המצב האופטימלי, כפי שקיים במעבד המקורי.

בעיה אפשרית: ישנן תוכניות בהן ניתן להגיע לפקודת ה- branch ממקומות שונים, ע"י קפיצה לכתובת פקודת ה- branch. במקרה כזה נצטרך לשמור שורות נוספות ב-BTB עבור כל פקודה שיכולה להיות הפקודה שלפני פקודת ה branch.

פתרון מספר 2:
בנוסף לכתובת הקפיצה, נשמור גם את הפקודה שנמצאת בכתובת הקפיצה, ובכך נחסוך את שלב ה- IF של הפקודה.

פתרון מספר 3
ניתן להוסיף חומרה בעזרתה נוכל להקדים את החיפוש ב- BTB לשלב ה- IF של פקודת ה- branch (כמו במעבד המקורי)

שאלה ממבחן

נתון מעבד הדומה ל-MIPS שנלמד בכיתה. במעבד זה הכרעת הקפיצה מתבצעת בשלב EX. המעבד מצויד במנגנון BTB והחיפוש בו מתבצע בשלב IF.

ה-BTB מסודר לפי כתובתה של הוראת הקפיצה. זמן החיפוש ב-BTB הוא פחות ממחזור שעון יחיד. עבור כל הוראת קפיצה נשמרת ב-BTB כתובת היעד ומכונת מצבים לחיזוי. ה-BTB אינו מכיל כל מידע נוסף.

נתון קטע הקוד הבא:

```
place2:  ADDI   R4,R4,#2
         ADDI   R4,R4,#2
         LD     R10,R4(128)
         ADDI   R9,R9,#0
         BEQ    R10,R1  place1
         ADDI   R7,R7,#1
place1:  BNEQ   R10,R1  place2
         ADDI   R8,R8,#1
```

ידוע כי לפני הרצת הקוד ה-BTB כבר הכיל את שתי הוראות הסיעוף המותנה המוצגות ואת כתובות היעד שלהן.

שאלה ממבחן (2)

בהנחה שהחיזויים של הוראות הסיעוף (BEQ ו- BNEQ) מדויקים, ושהוראת ה-BNEQ קופצת. כמה מחזורים יעברו מרגע שההוראה הראשונה כבר נמצאת בשלב IF ועד שהיא חוזרת אליו שוב?

```
place2:  ADDI   R4,R4,#2
         ADDI   R4,R4,#2
         LD     R10,R4(128)
         ADDI   R9,R9,#0
         BEQ    R10,R1  place1
         ADDI   R7,R7,#1
place1:  BNEQ   R10,R1  place2
         ADDI   R8,R8,#1
```

- BTB at IF (target + Bimodal)
- TKN/NTKN @ EXE

- Correct prediction of BEQ and BNEQ
- BNEQ is taken

- How many cycles for 1st place2 iteration?

<u>פתרון</u>: אם BNEQ קופצת, אז BEQ לא קופצת (התנאים שלהן הפוכים והפקודה בינהן לא משנה לא את R1 ולא את R10)

🡸סה"כ יש 7 פקודות באיטרציה => 7 מחזורי שעון

שאלה ממבחן (3)

ידוע כי למעבד נדרשו 33 מחזורי שעון לביצוע הקוד (כלומר, מרגע שההוראה הראשונה בקוד נמצאת בשלב IF ועד שההוראה האחרונה עזבה את הצינור-pipeline). כמה פעמים התבצעה הוראת LD?

```
place2:   ADDI   R4,R4,#2
          ADDI   R4,R4,#2
          LD     R10,R4(128)
          ADDI   R9,R9,#0
          BEQ    R10,R1  place1
          ADDI   R7,R7,#1
place1:   BNEQ   R10,R1  place2
          ADDI   R8,R8,#1
```

- BTB at IF (target + Bimodal)
- TKN/NTKN @ EXE

- Correct prediction of BEQ and BNEQ
- BNEQ is taken

- How many LD was executed in 33 cycles ?

<u>פתרון:</u>

- באיטרציה אחרונה BNEQ לא קופצת ← BEQ קופצת
- גם באיטרציה האחרונה יש 7 פקודות
- על כל טעות בחיזוי משלמים 2 מחזורי שעון. למילוי pipeline צריכים 4 מחזורי שעון.

סה"כ התכנית לוקחת:

7*#iterations + 2*#branch_prediction_errors + 4= 33

בגלל שבכל איטרציה יש 2 פקודות branch,

2*#iterations >= #branch_prediction_errors

הפתרון היחיד:

#iterations=3, #branch_prediction_errors = 4

=> פקודת LD מתבצעת 3 פעמים.

שאלה ממבחן (4)

תחת ההנחות של הסעיף הקודם, כמה פעמים היה החיזוי להוראות הסיעוף נכון, וכמה פעמים שגוי?

<u>פתרון:</u> ישנן 3 איטרציות ← 6 פעמים מתבצע branch
לפי סעיף ב' יש 4 טעויות חיזוי ← 2 חיזויים נכונים.

תן דוגמא למכונת מצבים שתתאים לתוצאה שהשגת בסעיף הקודם. עליך לציין את מצב המכונה של כל אחת מההוראות לפני ביצוע הקוד.

<u>פתרון</u>: מכונה של 2 מצבים (מצב התחלתי של BEQ– T ושל BNEQ– NT):