מבוא למחשבים

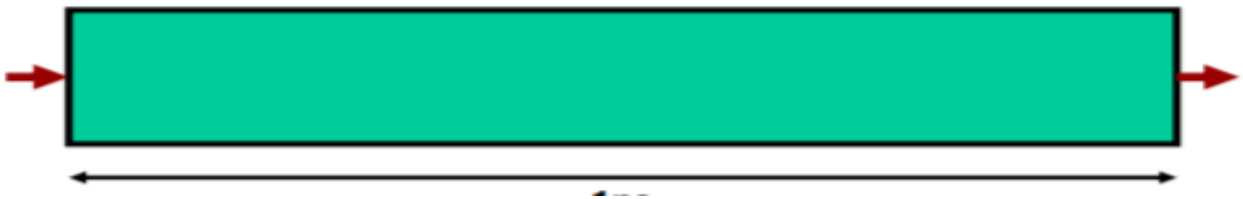
Pipeline Processing

השקפים מבוססים על הספרים
Mano – פרק 9
Hennessy & Patterson

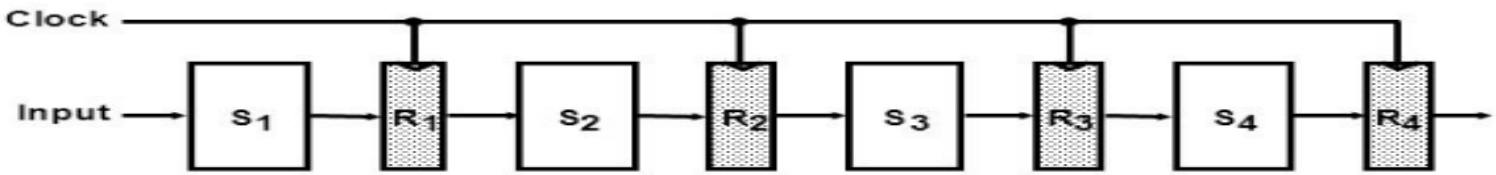
ד"ר רון שמואלי
rshmueli@bgu.ac.il

Pipeline Processing

- Key idea: break big computation into pieces



General Structure of a 4-Segment Pipeline



Space-Time Diagram

Segment	1	2	3	4	5	6	7	8	9
1	T1	T2	T3	T4	T5	T6			
2		T1	T2	T3	T4	T5	T6		
3			T1	T2	T3	T4	T5	T6	
4				T1	T2	T3	T4	T5	T6

Clock cycles

חישוב גורם האצה של ה- PIPELINE

- **n -מספר המשימות לביצוע/ K - מספר הסיגמנטים.**
- **במחשב ללא Pipelined.**
  - T- הזמן להשלים משימה.
  - n*T הזמן הנדרש להשלמת n משימות.
- **במכונת Pipelined**
  - tp – זמן מחזור של השעון (זמן לסיום כל סיגמנט)
  - Ktp – זמן להשלמת המשימה הראשונה.
  - (n-1)tp - הזמן להשלמת n-1 המשימות הנותרות.
  - (k+n-1)tp – הזמן להשלמת n משימות.
- **גורם ההאצה Speedup**

$$Sk = n*T / (k + n - 1)*tp$$

תוצאות מיטביות

ה- PIPELINE תמיד מלא
גורם האצה תיאורטי הוא K
(K מספר הסגמנטים)

$$\lim_{n \to \infty} S_k = \frac{T}{t_p} = k, \quad (\text{ if } t_n = k * t_p)$$

בהנחה $T=kt_p$

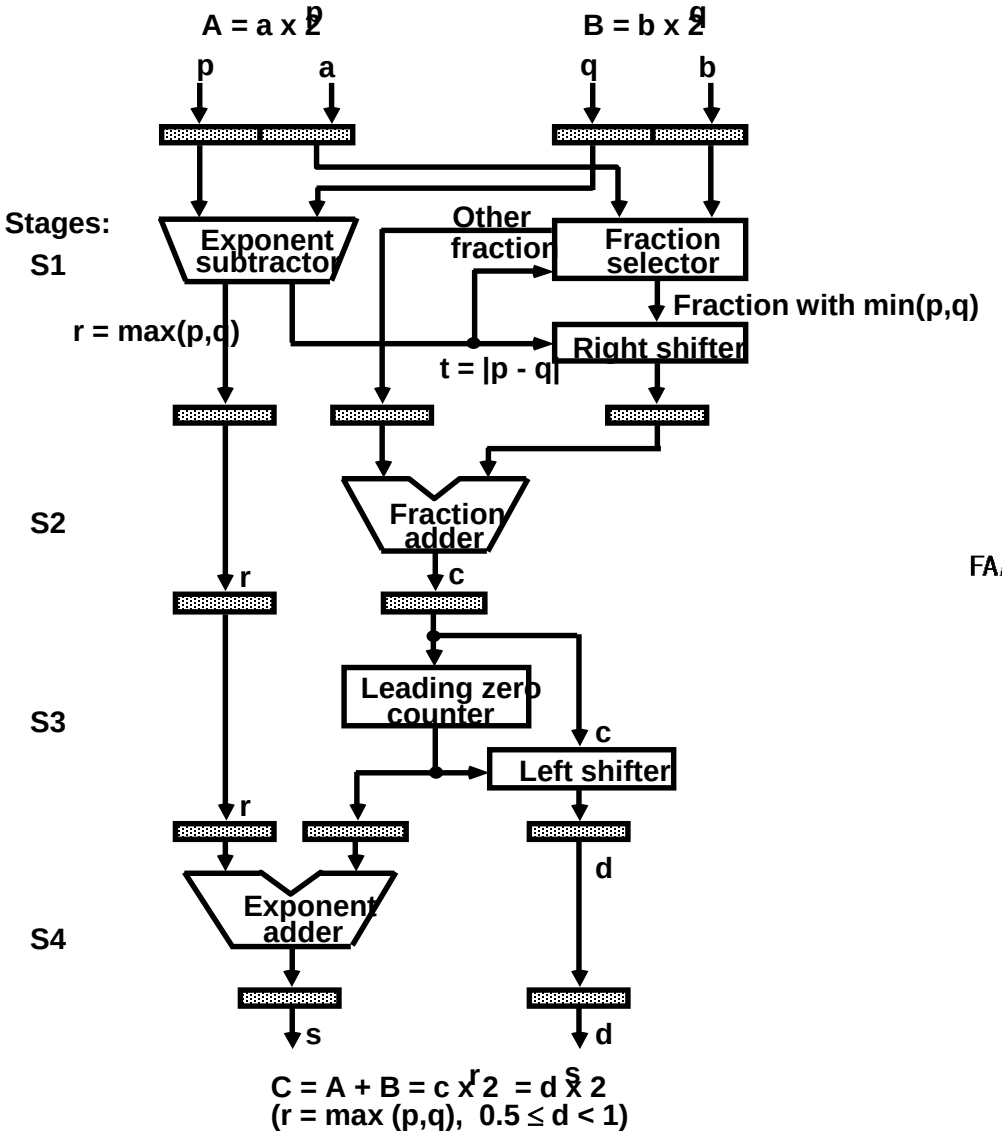
הגבול המעשי של Pipeline

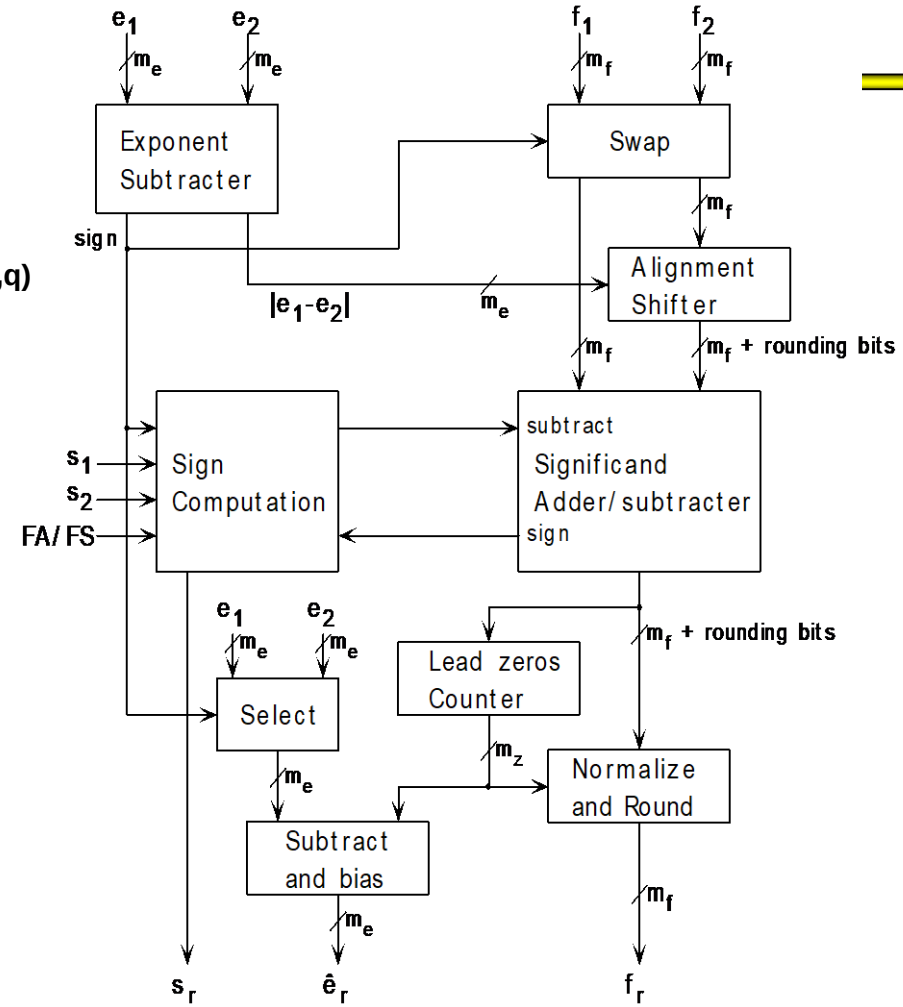
- Pipeline - **לא יכול להגיע למהירות התאורטית**

  – סיגמנטים שונים דורשים זמן ביצוע שונה ← תאום תדר שעון לסגמנט האיטי← זמן המתנה בסגמנטים האחרים.
  - בזבוז זמן עם אוגרי הביניים (לא נדרשים במערכת רגילה).

- Pipeline– טוב לשימוש:
  - פעולות אריתמטיות
  - ביצוע במקביל של מחזור פקודה (fetch-decode-execute)

4-STAGE FLOATING POINT ADDER





חישוב גורם האצה

- **בהנחה שהזמן הנדרש לסיגמנטים השונים הוא:**
  - t1=60ns, t2=70ns, t3=100ns, t4=80ns
  - ההשהיה של האוגרים tr=10ns
- זמן מחזור (כל מחזור מתקבלת תפוקה):

tp = t3+tr=100+10=110

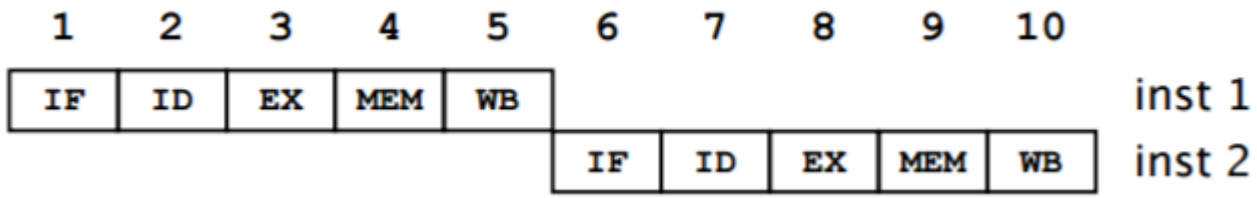
- זמן ביצוע ללא Pipeline (סכום הזמנים):

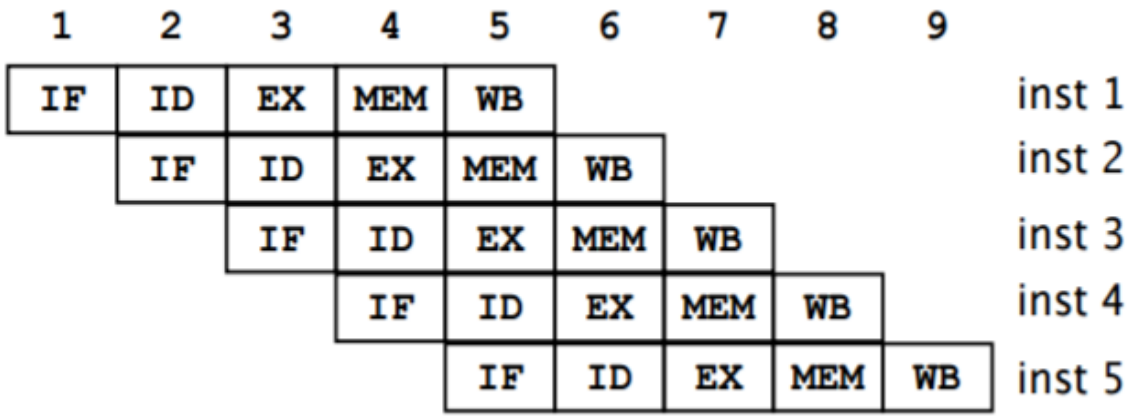
tn = t1+ t2+ t3+t4+tr= 60+70+100+80ns= 320

- S=tn/tp=320/110=2.9

CPU cycle


Instruction Fetch, IF
Decode, D
Read Regs, R
Execute, E
Data Fetch, DF
Write Reg, W

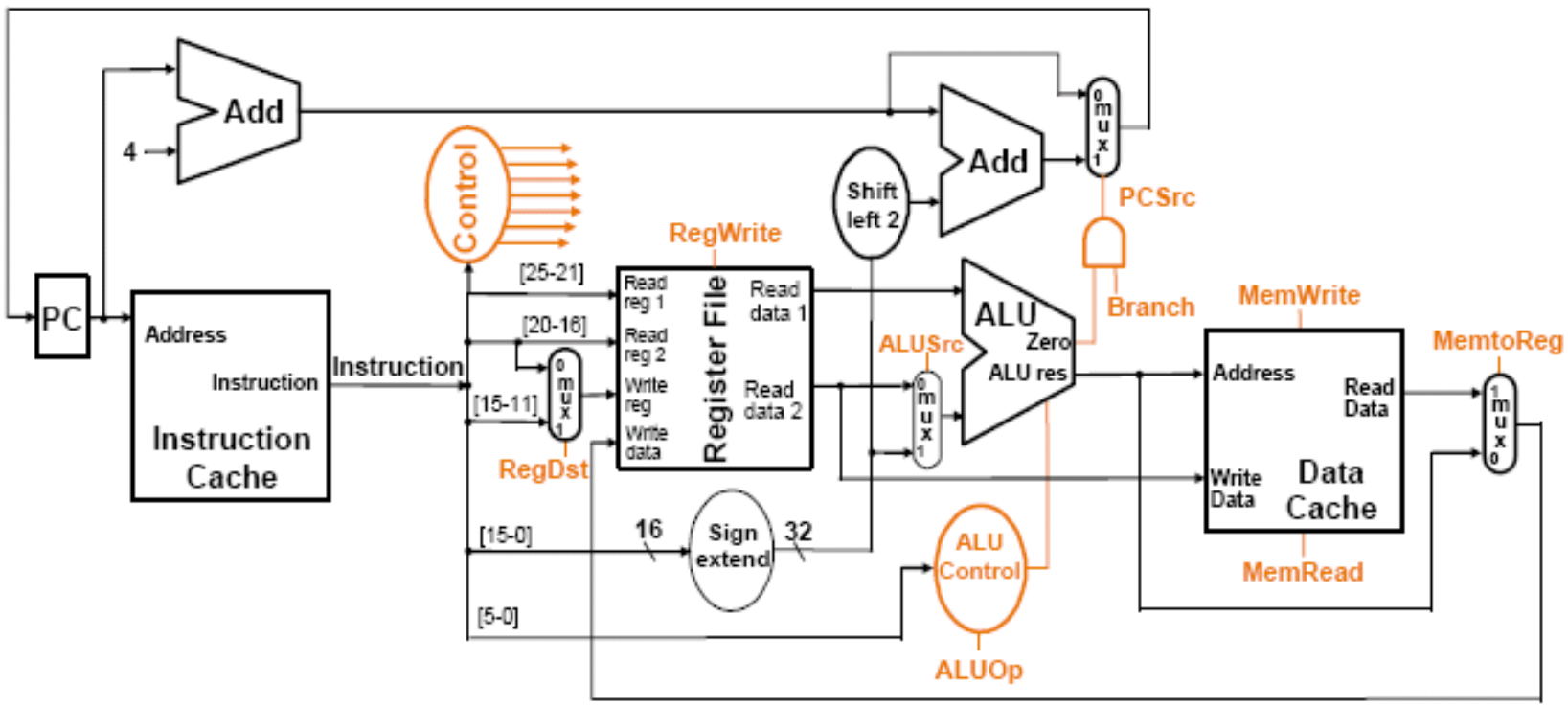

1 2 3 4 5 6 7 8 9 10
IF ID EX MEM WB
inst 1
IF ID EX MEM WB
inst 2


1 2 3 4 5 6 7 8 9
IF ID EX MEM WB
inst 1
IF ID EX MEM WB
inst 2
IF ID EX MEM WB
inst 3
IF ID EX MEM WB
inst 4
IF ID EX MEM WB
inst 5

The MIPS CPU

MIPS - Microprocessor without Interlocked Pipeline Stages

Instruction fetch	Instruction Decode / register fetch	Execute / address calculation	Memory access	Write back



Hennessy & Patterson - Computer Architecture

MIPS Instruction Formats

- R-type (register insts)

31–26	21–16	16–11	11–6	6–0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- I-type (Load, Store, Branch, inst's w/imm data)

31–26	21–16	16–0
op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

- J-type (Jump)

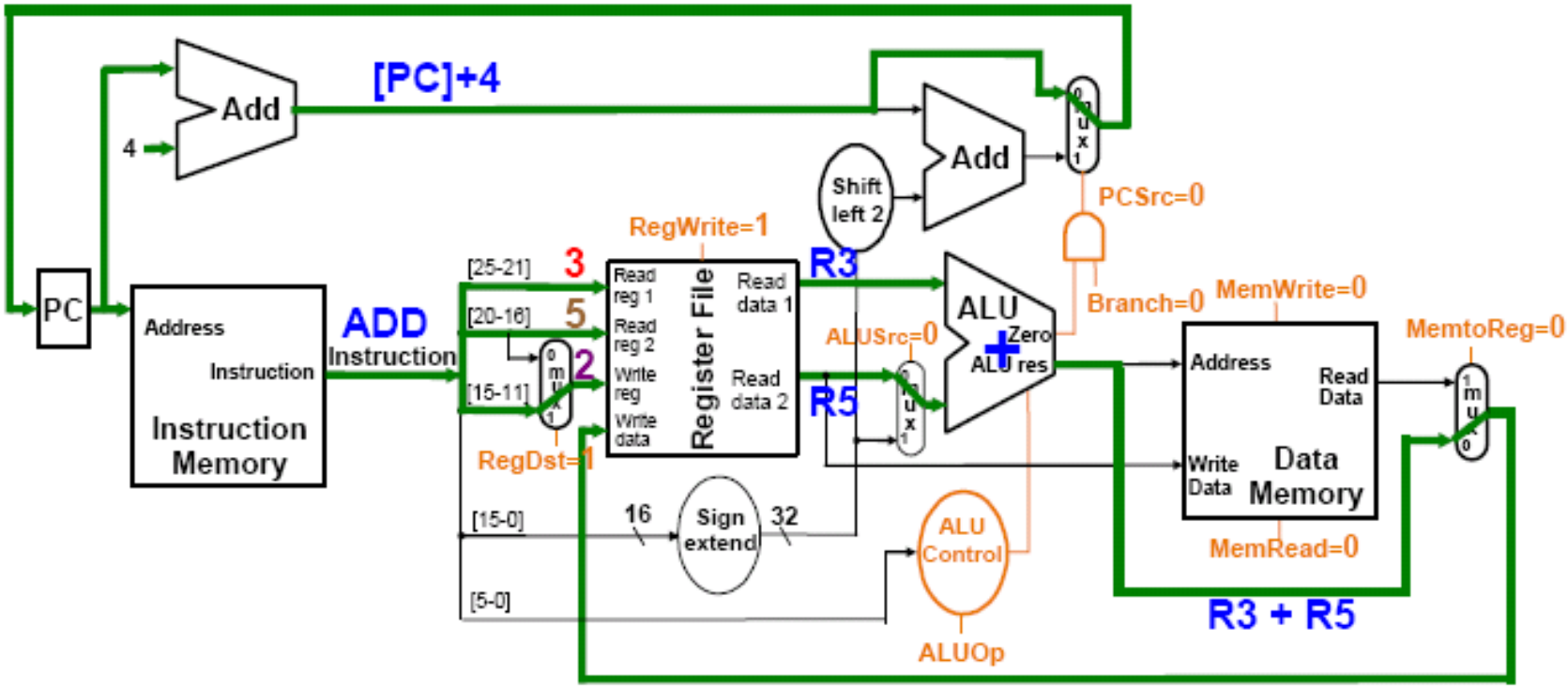
31–26	0
op	target address
6 bits	26 bits

op: operation of the instruction
rs, rt, rd: the source and destination register specifiers
shamt: shift amount
funct: selects the variant of the operation in the "op" field
address / immediate: address offset or immediate value
target address: target address of the jump instruction

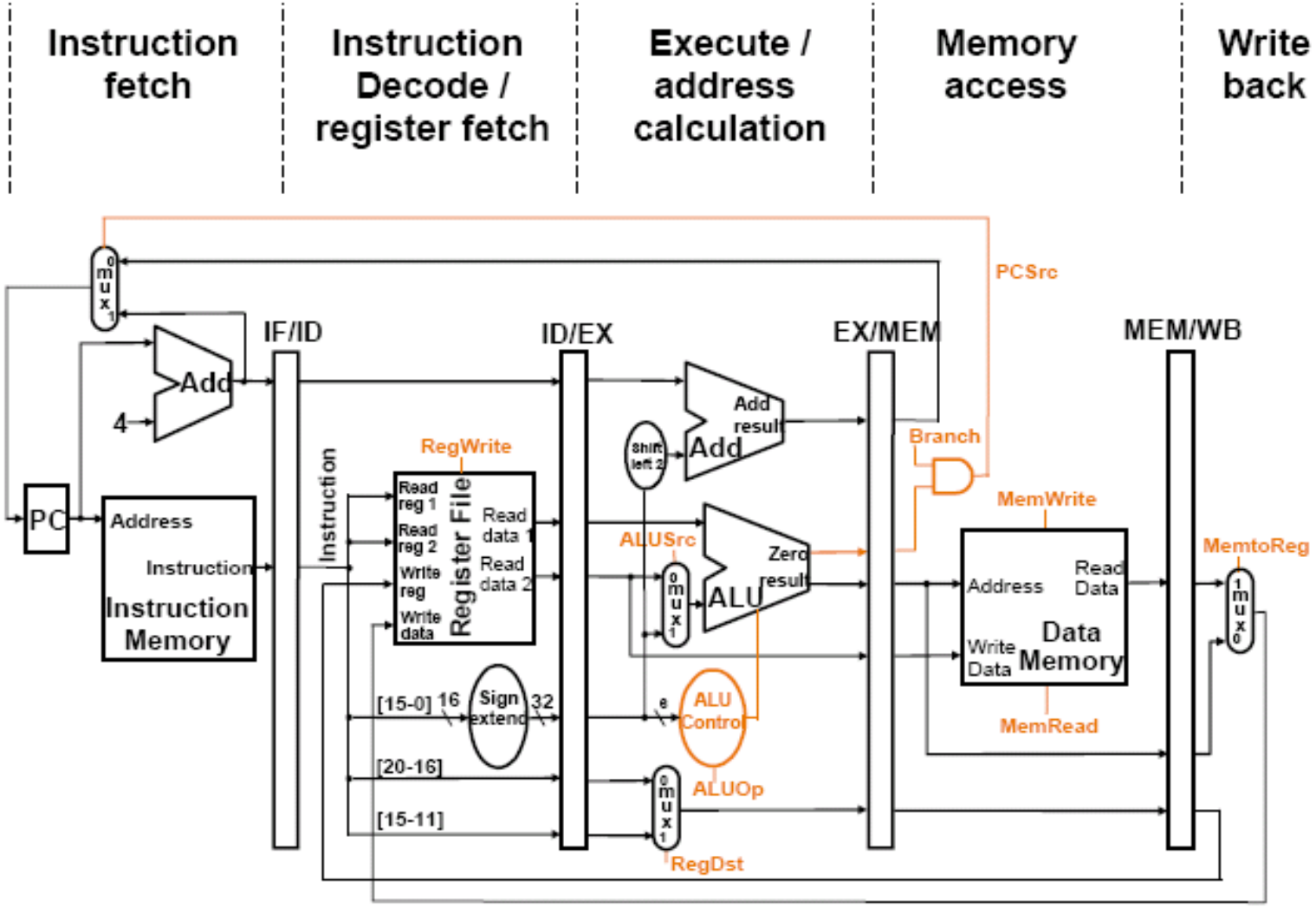
Executing an Add Instruction

Add R2, R3, R5 ; R2 ← R3+R5

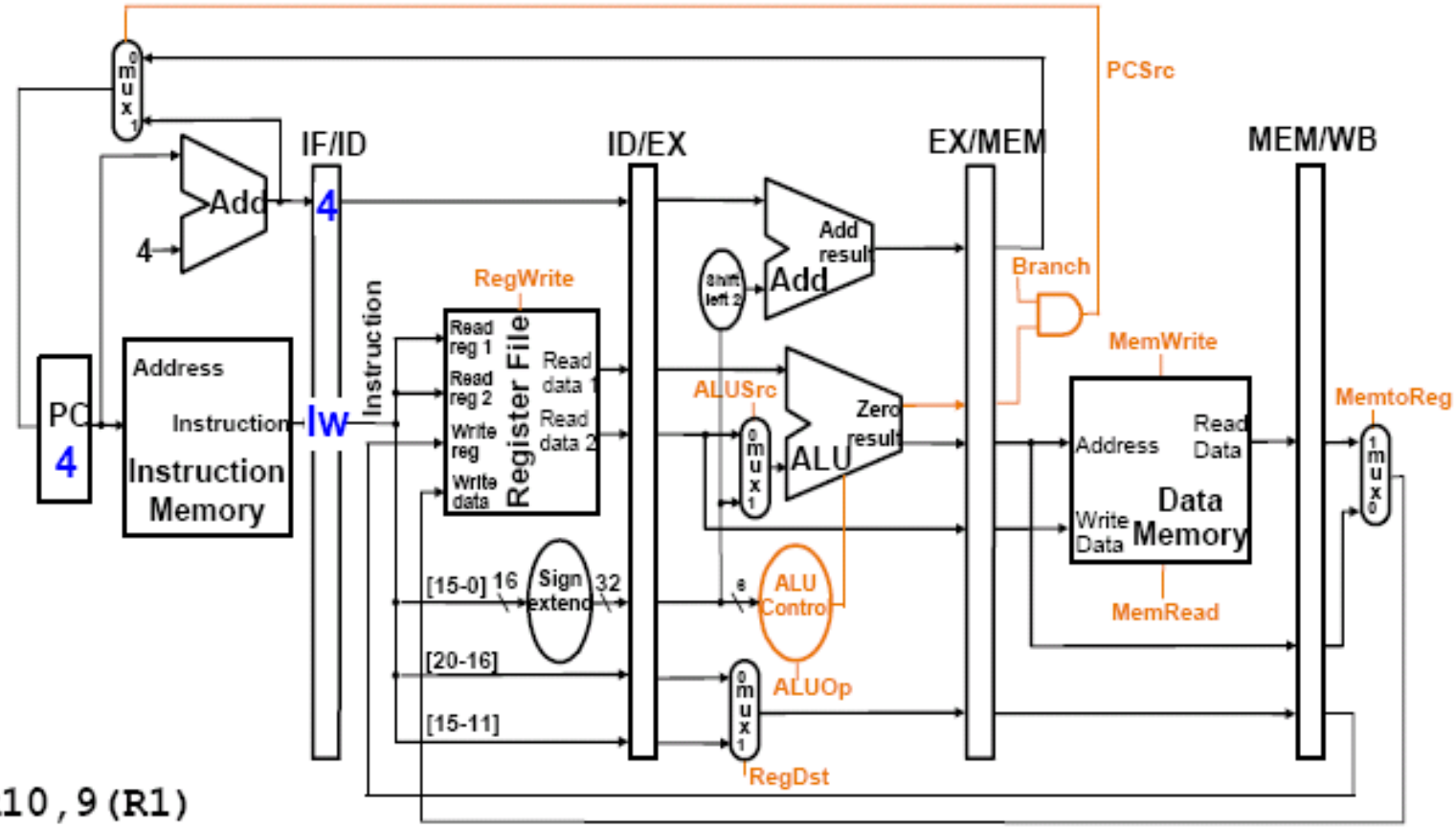
31 op 26	rs 21	rt 16	rd 11	shamt 6	funct 0
ALU	3	5	2	0	0 = Add



Pipelined CPU

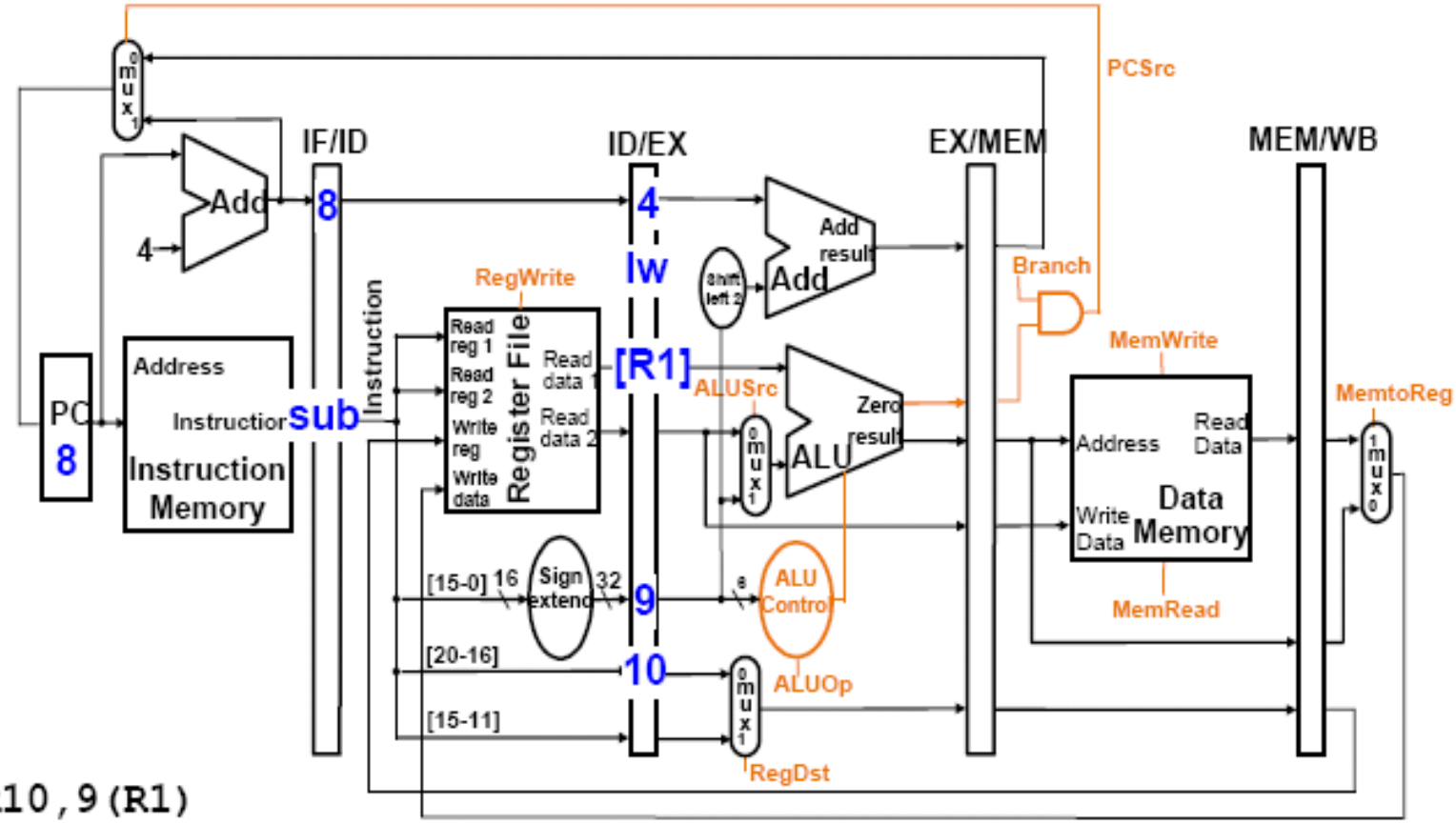


Pipeline Example: cycle 1



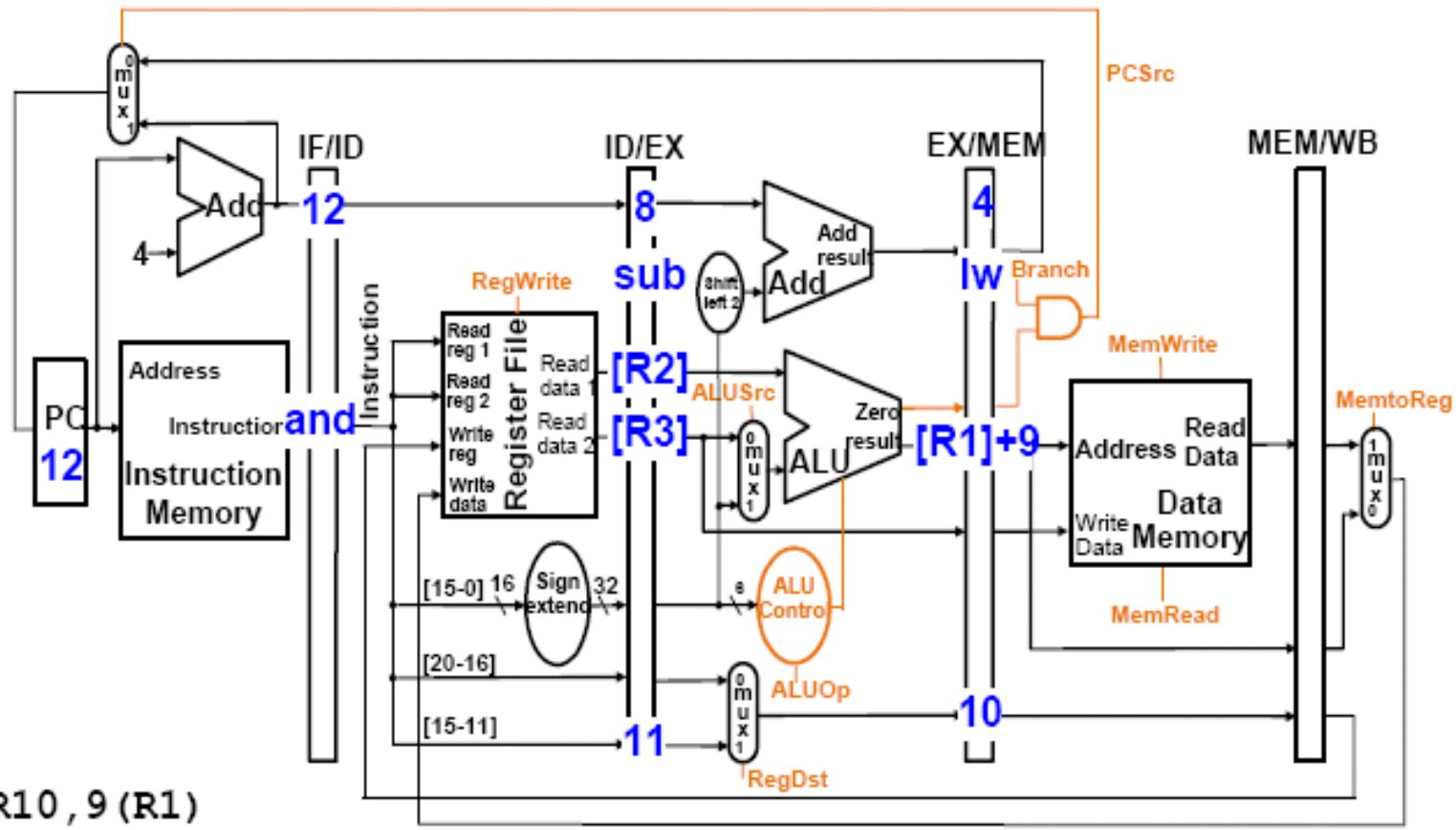

```
lw  R10,9(R1)
sub R11,R2, R3
and R12,R4, R5
or  R13,R6, R7
```

Pipeline Example: cycle 2



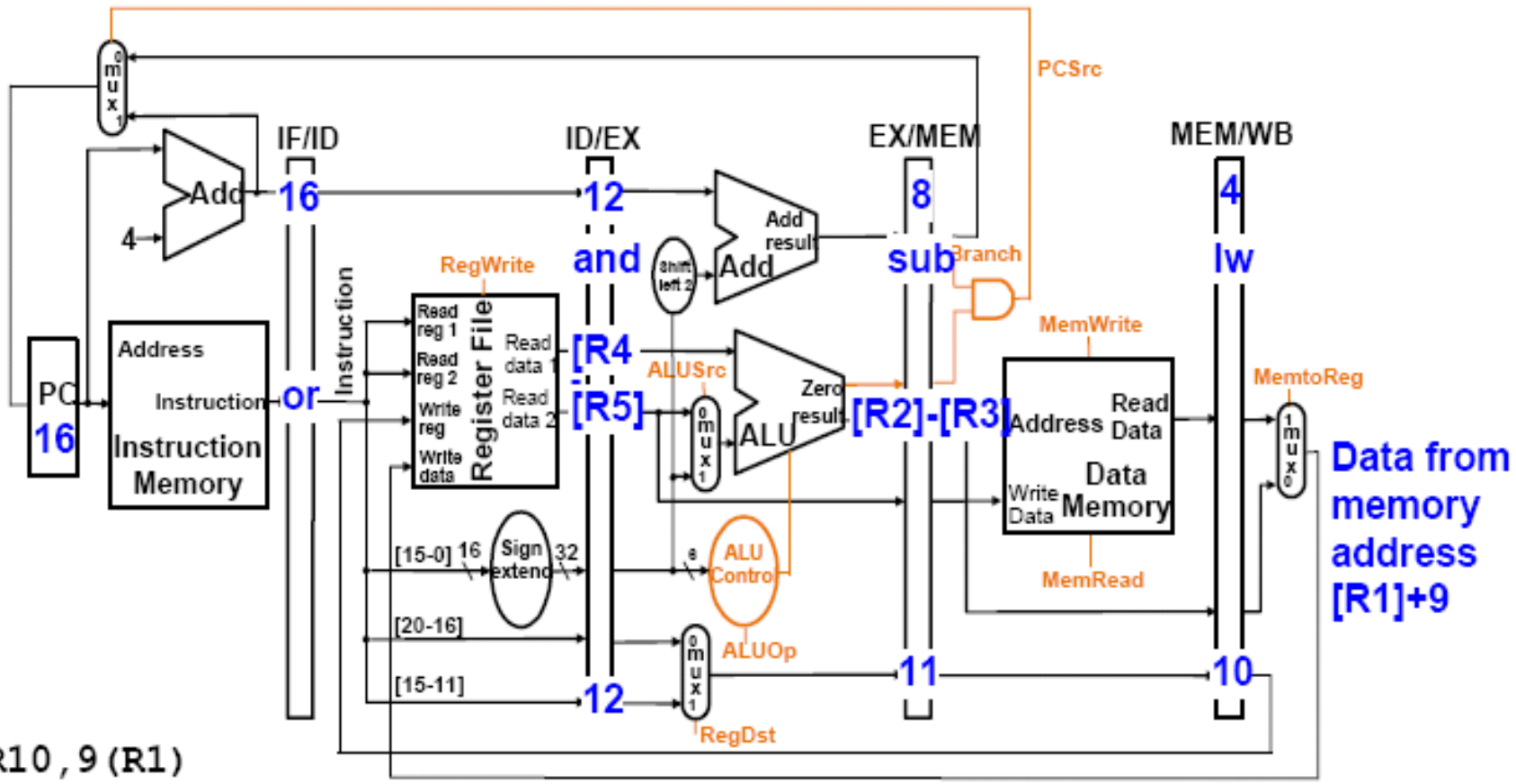

```
lw  R10,9(R1)
sub R11,R2, R3
and R12,R4, R5
or  R13,R6, R7
```

Pipeline Example: cycle 3




```
lw  R10,9(R1)
sub R11,R2, R3
and R12,R4, R5
or  R13,R6, R7
```

Pipeline Example: cycle 4



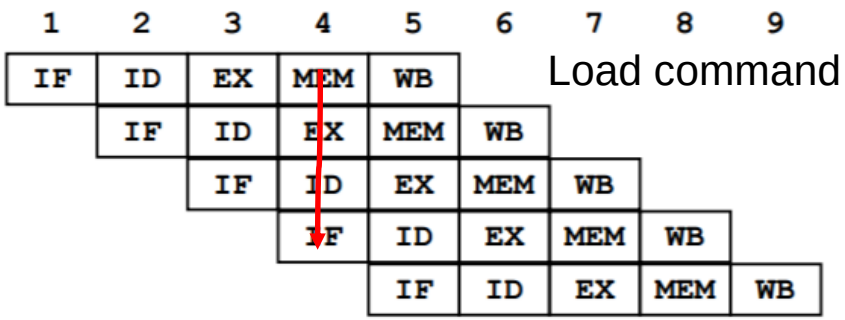

```
lw  R10,9(R1)
sub R11,R2, R3
and R12,R4, R5
or  R13,R6, R7
```

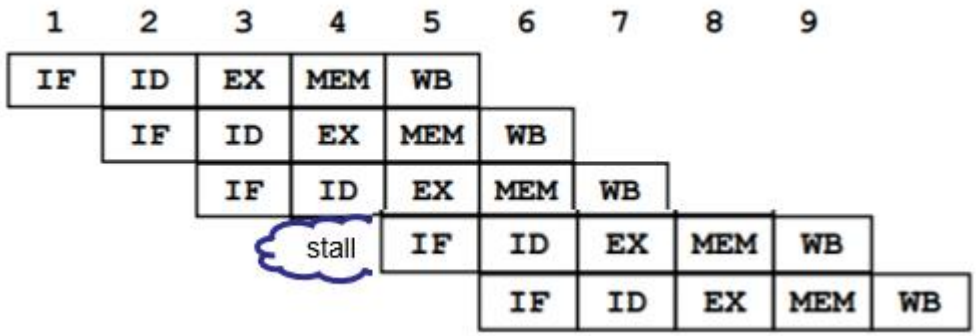
סיכונים ב- Pipeline - תקציר

- **Structural hazards** - גישה לזיכרון ע"י שני סגמנטים.
- **חציית אוגרים** – גישה לאוגרים לקריאה וכתיבה.
- **Data hazards** - תלות בין הנתונים המתבצעים.
  - **Hardware interlock**– מעגל המשהה את הפקודה
  - **Operand forwarding** – ניתוב תוצאה במסלול חלופי .
  - **Delayed load** -_הקומפיילר מסדר את הפקודות מחדש
- **Control Hazards -** בקרת זרימה-הסתעפות.

Structural Hazards

– גישה לזיכרון ע"י שתי בפקודות, בשלבי ביצוע שונים.





What if Instruction 1 is also a LOAD?

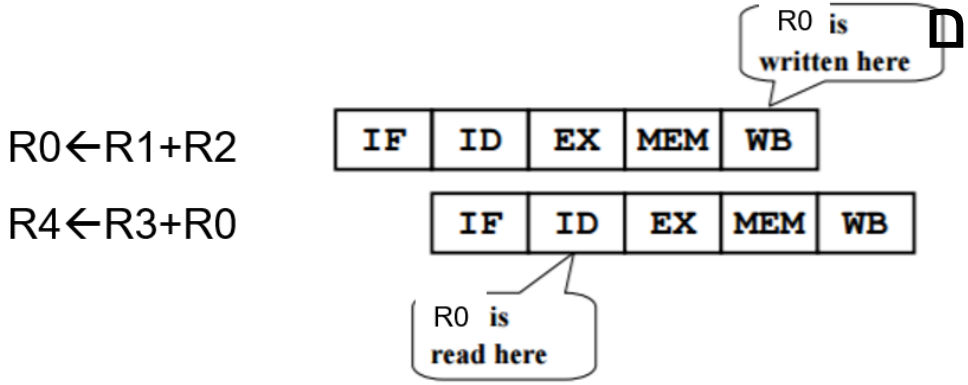
פתרונות

– הכנסת STALL

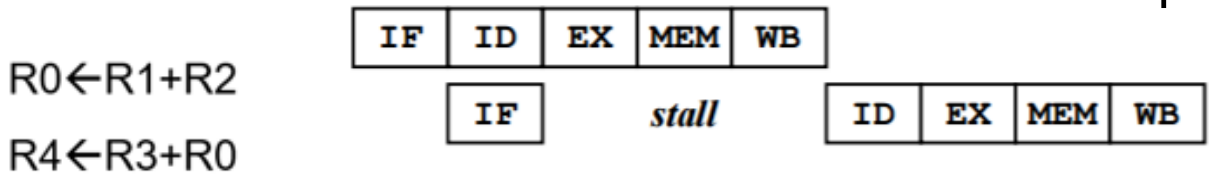
– הפרדת זיכרונות

Data Hazards (1)

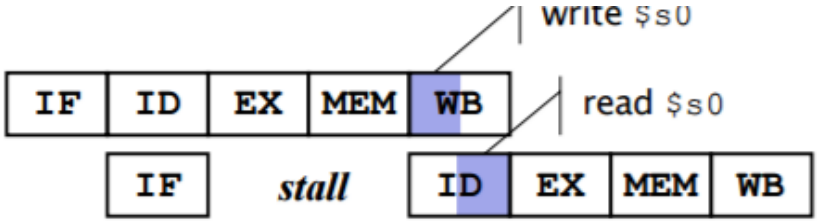
- תלות בין הנתונים המתבצעים



- **Hardware interlock**
  - הכנסת Stalls עד שהפקודה זמינה

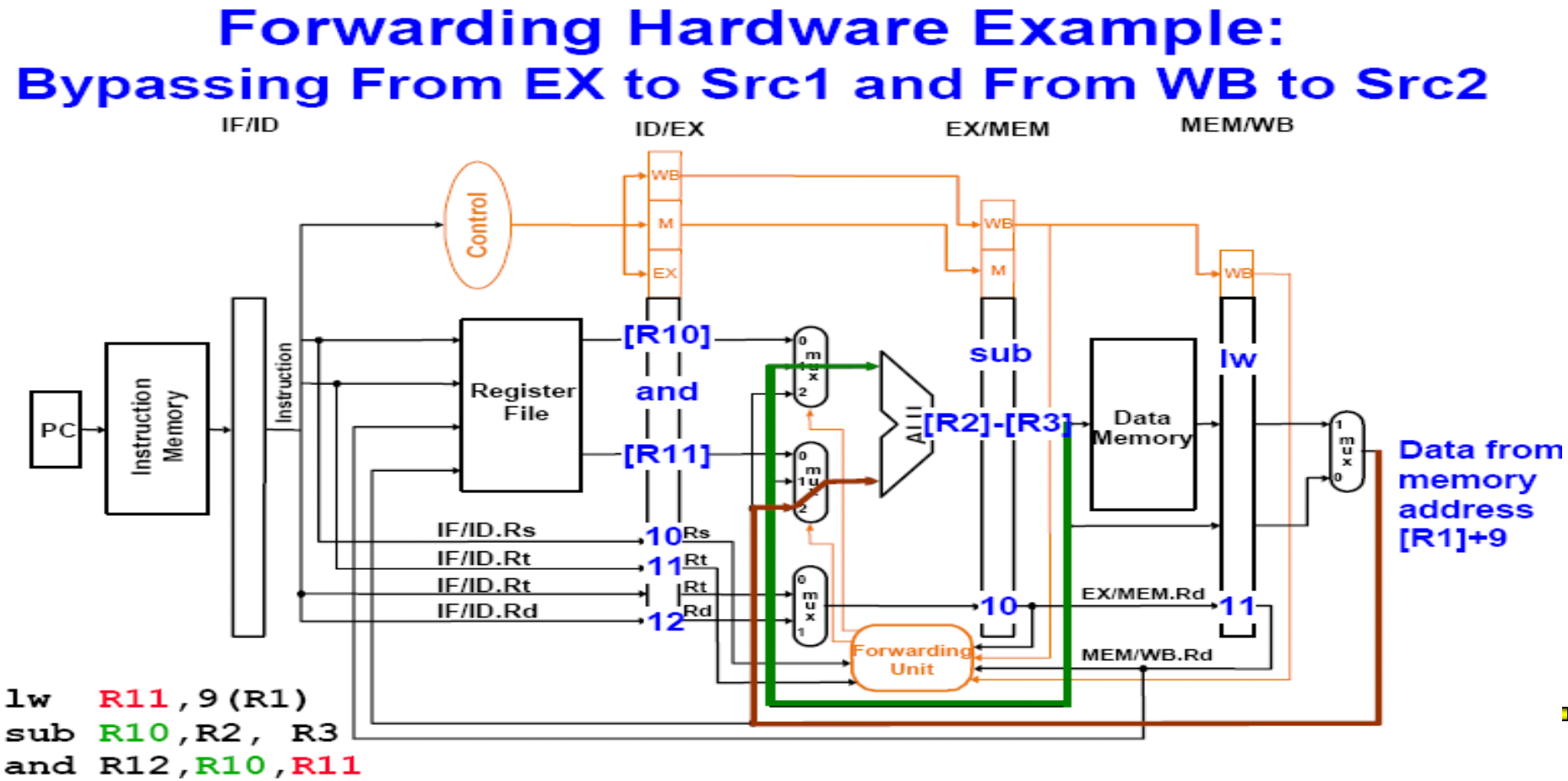


- **חציית אוגרים**
  - כתוב לרגיסטר בחצי שעון ראשון, קרא בחצי שעון שני



Data Hazards – Operand Forwarding

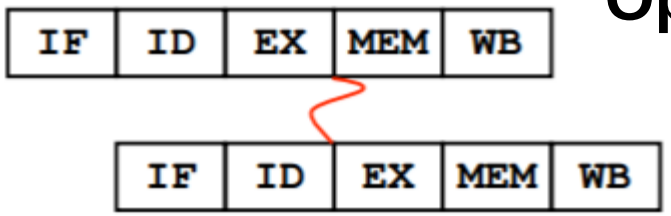
- חומרה שמגלה תלות בין פקודות, ומנתבת תוצאה במסלול חלופי ישירות לסגמנט המתאים.



דוגמא

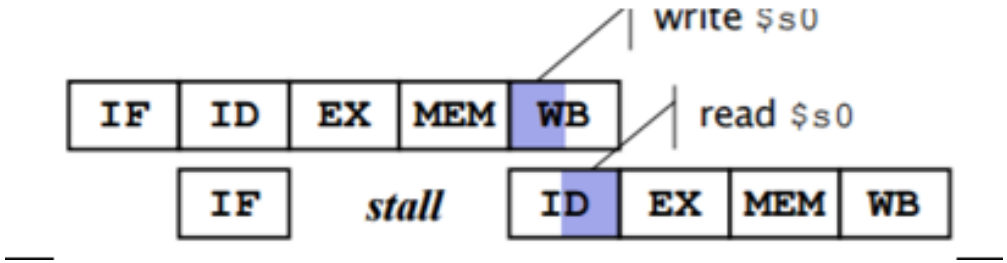
R0←R1+R2

R4←R3+R0



Operand Forwarding

ביחס ל HW Interlock



Data Hazard- Delayed Load

- הקומפיילר מסדר את הפקודות מחדש במידת הצורך מכניס NOPs

דוגמא - Delayed Load+ Forwarding

a = b + c;
d = e - f;

Unscheduled code:

```
    ld      r2, b(0)
    ld      r3, c(0)
→   add     r1, r2, r3
→   st      r1, a(0)
    ld      r5, e(0)
    ld      r6, f
→   sub     r4, r5,r6
→   st      r4,d
```

סידור פקודות מחדש ע" המהדר

Scheduled Code:

```
ld      r2, b(0)
ld      r3, c(0)
ld      r5, e(0)
add     r1, r2, r3
ld      r6, f
st      r1,a(0)
sub     r4, r5,r6
nop (or stalls by the HW)
st      r4,d(0)
```

Delayed Load - A load requiring that the following instruction not use its result

Data hazards – HW interlock

Time (in clock cycles) →

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2:	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM:	X	X	X	–20	X	X	X	X	X
Value of MEM/WB:	X	X	X	X	–20	X	X	X	X

Program execution order (in instructions)

עם חציית אוגרים

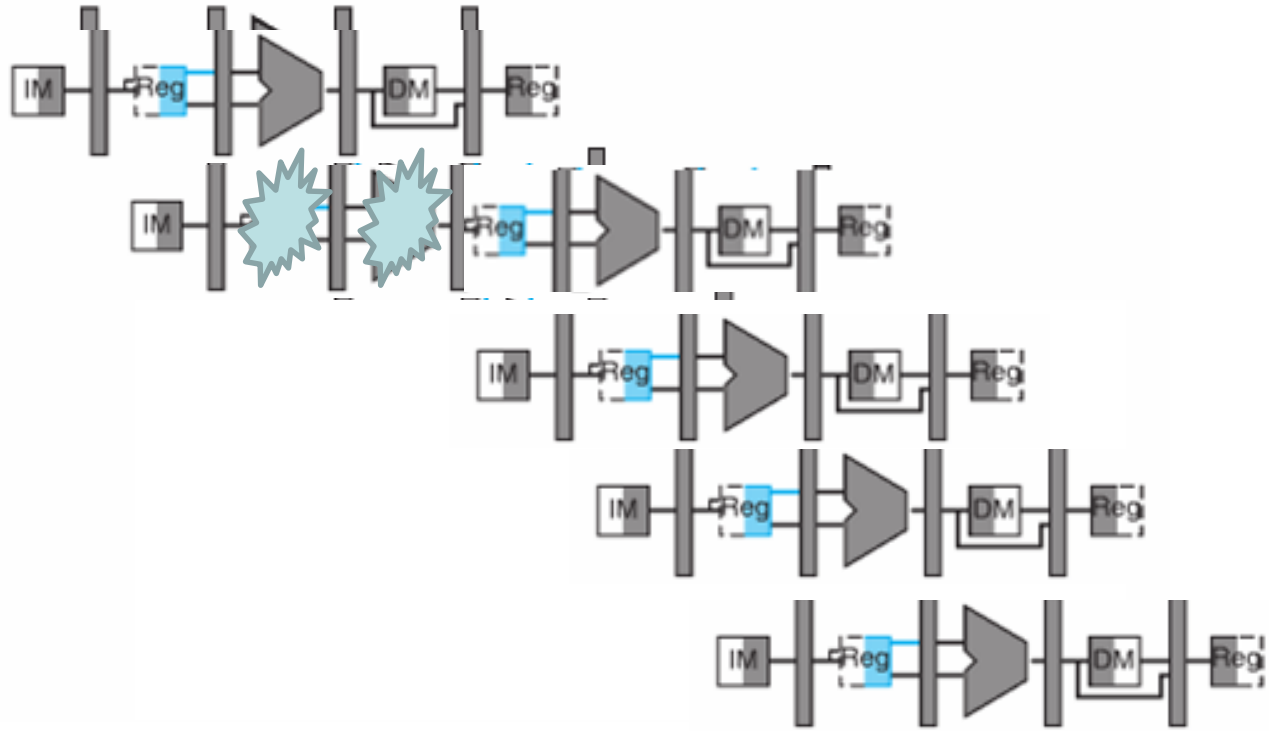
R2←R1-R3

R12← R2 and R5

R13← R6 or R2

R14←R2+R2

M[R2+100] ← R15



Data hazards - Forwarding

Time (in clock cycles)	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2:	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM:	X	X	X	–20	X	X	X	X	X
Value of MEM/WB:	X	X	X	X	–20	X	X	X	X

עם FORWARDING + חציית אוגרים

Program execution order (in instructions)

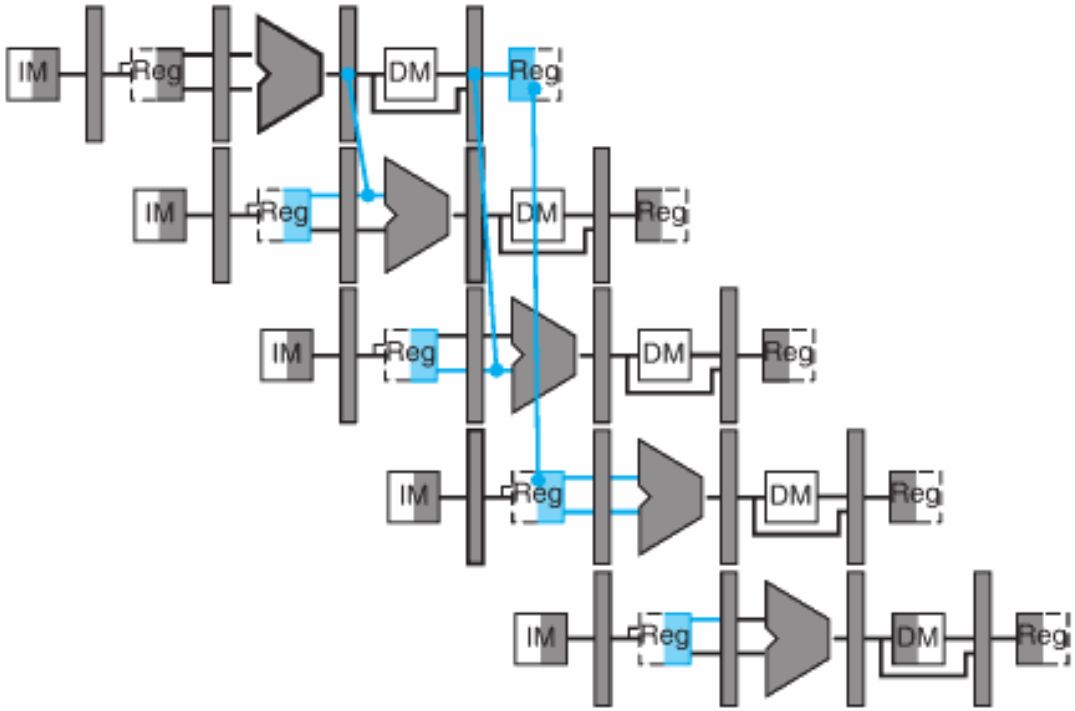
R2←R1-R3

R12← R2 and R5

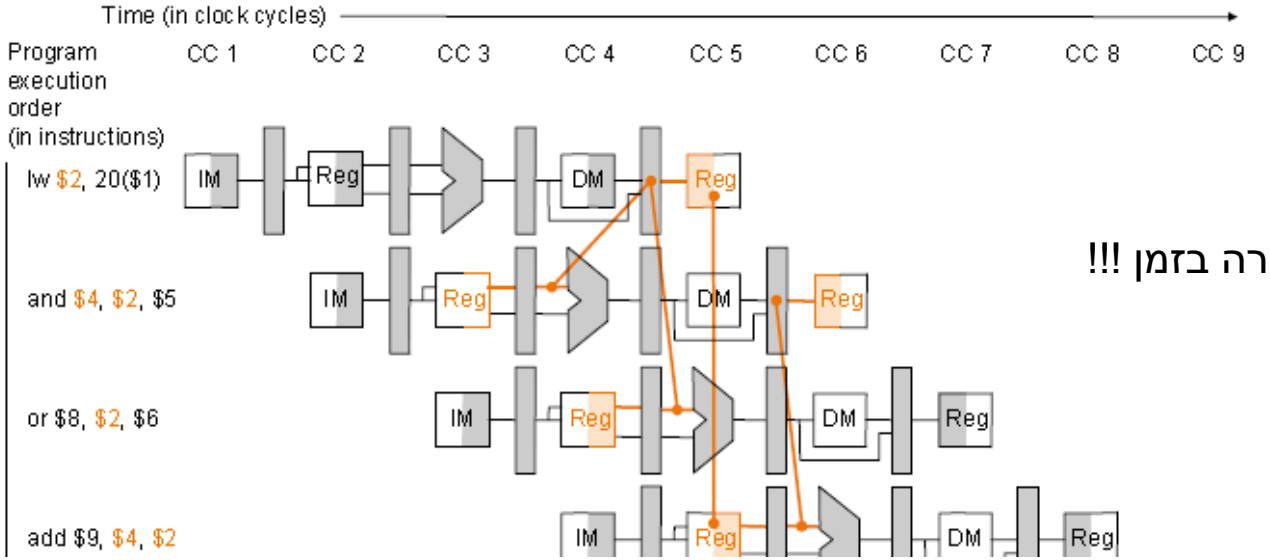
R13← R6 or R2

R14←R2+R2

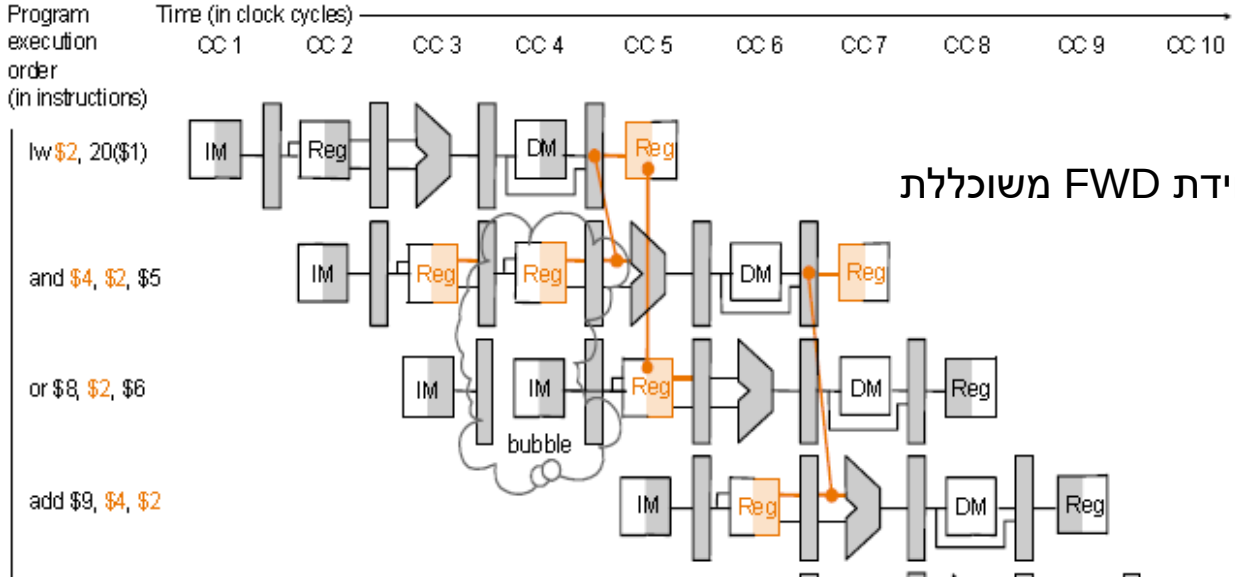
M[R2+100] ← R15



Data Hazards that cannot solved by FWD



נדרש חזרה בזמן !!!



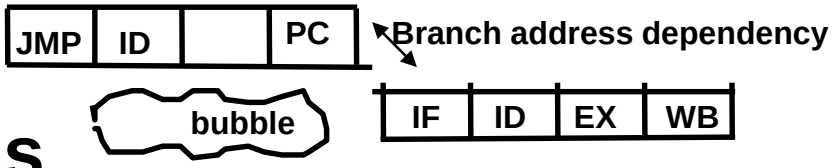
Stall הכרחי גם עם יחידת FWD משוכללת

Control Hazards - בקרת זרימה-הסתעפות.

- **Branch Prediction**- ניסיון לנחש את תוצאת ההסתעפות – והבאת הפקודות המתאימות בהתאם לניחוש
- **Branch target Buffer (BTB)** (ניהול הזיכרון אסוציאטיבי)
- **Delayed Branch** - המהדר מסדר את הפקודות מחדש, מכניס פקודות שימושיות – כך שהצינור יישאר מלא כאשר ישנה הסתעפות
- ***PRE-FETCH*** *ביצוע FETCH לכתובת הבאה ולכתובת הסתעפות.*
- ***Loop Buffer (High Speed Register file)*** *– שמירת הלולאה כולה כולל ההסתעפות באוגרים הלולאה כולה מתבצעת ללא גישה לזיכרון.*

Control Hazard on Branches (1)

- **<u>Static Option 1: Stall</u>**
  - **Stall pipe when branch is encountered until resolved**



- **Stall impact: assumptions**
  - **CPI = 1**
  - **20% of instructions are branches**
  - **Stall 3 cycles on every taken branch**
- **CPI new = 1 + 0.2 × 3 = 1.6**
  - **(CPI new = CPI Ideal + avg. stall cycles / instr.)**
- **We loose 60% of the performance**

Control Hazard on Branches (2)

- **Static Option 2: Predict Not Taken**
  - **Execute instructions from the fall-through (not-taken), path**
    - **As if there is no branch**
    - **If the branch is not-taken (~50%), no penalty is paid**
  - **If branch actually taken**
    - **Flush the fall-through path instructions before they change the machine state (memory / registers).**
    - **Fetch the instructions from the correct (taken) path**
  - **Assuming ~50% branches not taken on average**
    - **CPI new = 1 + (0.2 × 0.5) × 3 = 1.3**

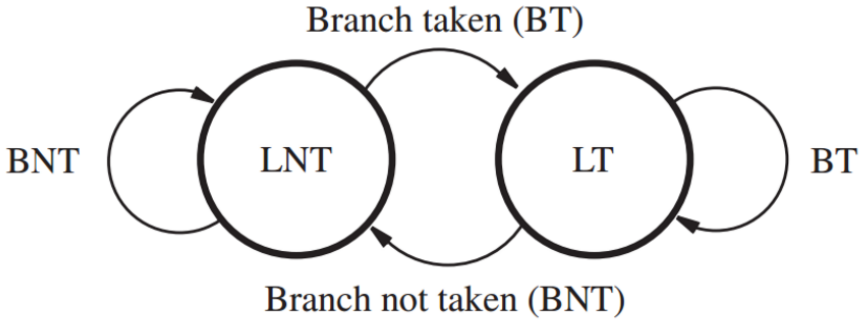
Branch Prediction

- חיזוי דינמי נעשה בהתאם להיסטוריה.

- טכניקה 1 – סיבית חיזוי בהתאם להסתעפות האחרונה (Taken/Not Taken).
  - נניח מצב התחלתי דגל חיזוי =Not Taken
  - בכניסה ללולאה Miss, דגל חיזוי = Taken
  - תשע פעמים חיזוי נכון (HIT)
  - בסוף הלולאה MISS דגל חיזוי = Not Taken)
- פגיעה ב- 8 מתוך 10

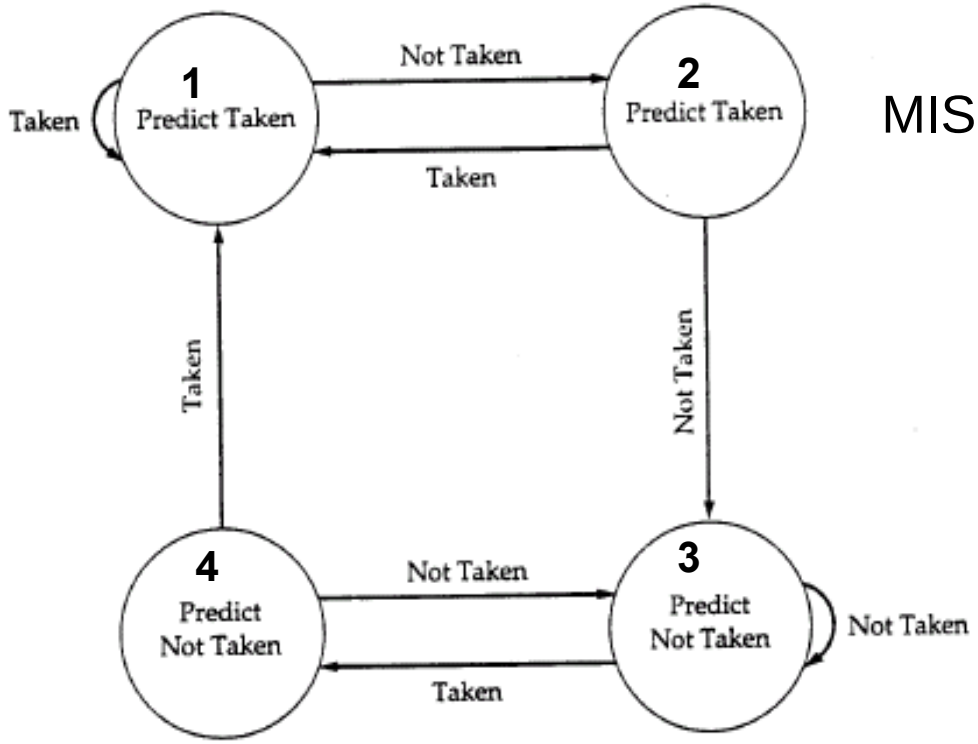
```
LOP  LDA  NUM
     ISZ  NUM
     BUN  LOP
     STA  STM
          ....
NUM  -10
STM    0
```

- בכניסה הבאה ללולאה MISS - דגל חיזוי =Not Taken



Branch Prediction

- **טכניקה 2** משתמשת בשתי סיביות – נועדה למנוע את ה- MISS בכניסה השנייה
- כיוון שנדרש MISS פעמיים
- בכניסה ללולאה פעם נוספת:
- התוכנית תהיה במצב 2 ולכן לא יהיה MISS

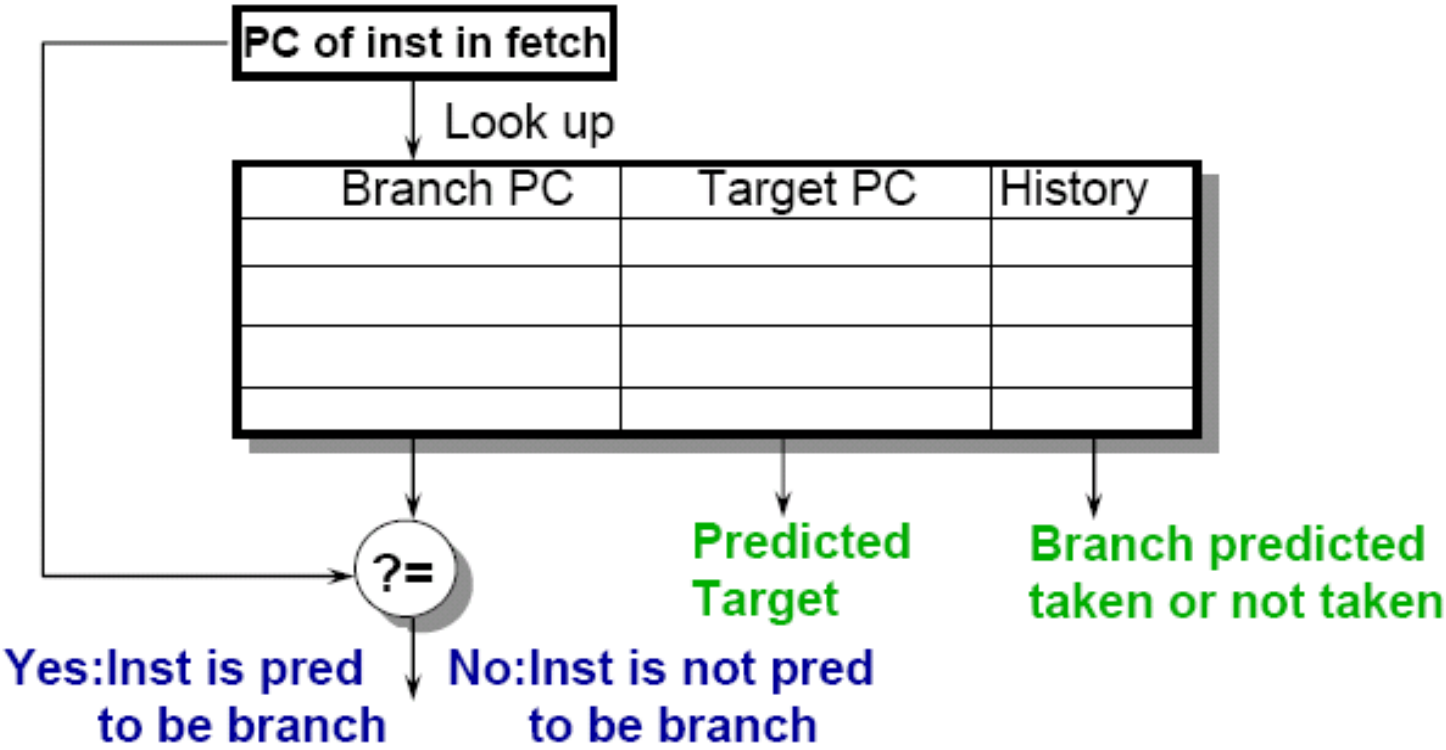



```
LOP  LDA  NUM
     ISZ  NUM
     BUN LOP
     STA  STM
          ....
NUM  -10
STM    0
```

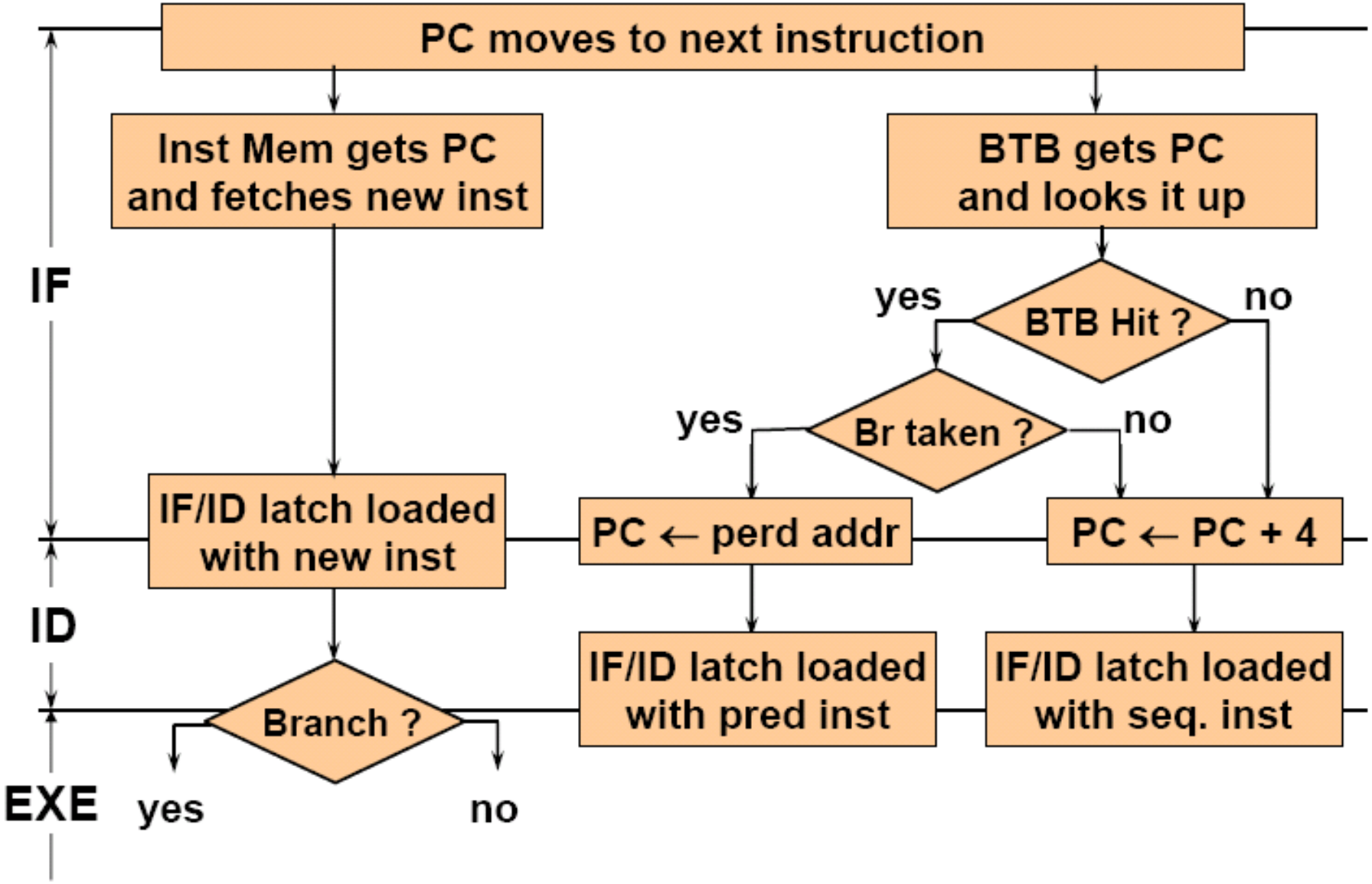
- טכניקה נוספת – תלויה גם בביצוע של ההסתעפות האחרונה שהתבצעה.

Dynamic Branch Prediction

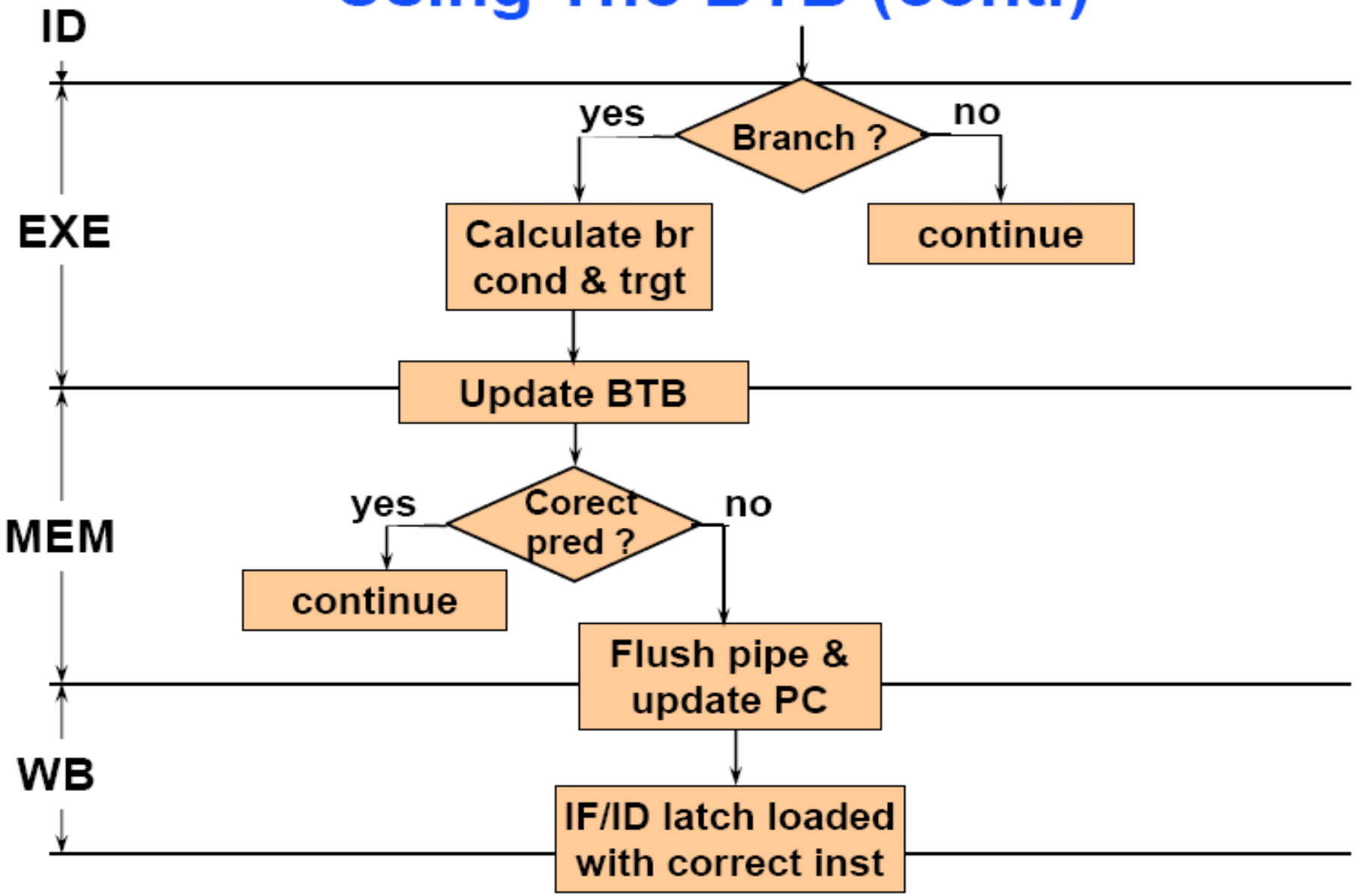
- Add a Branch Target Buffer (BTB) the predicts (at fetch)
  - Instruction is a branch
  - Branch taken / not-taken
  - Taken branch target



Using The BTB



Using The BTB (cont.)



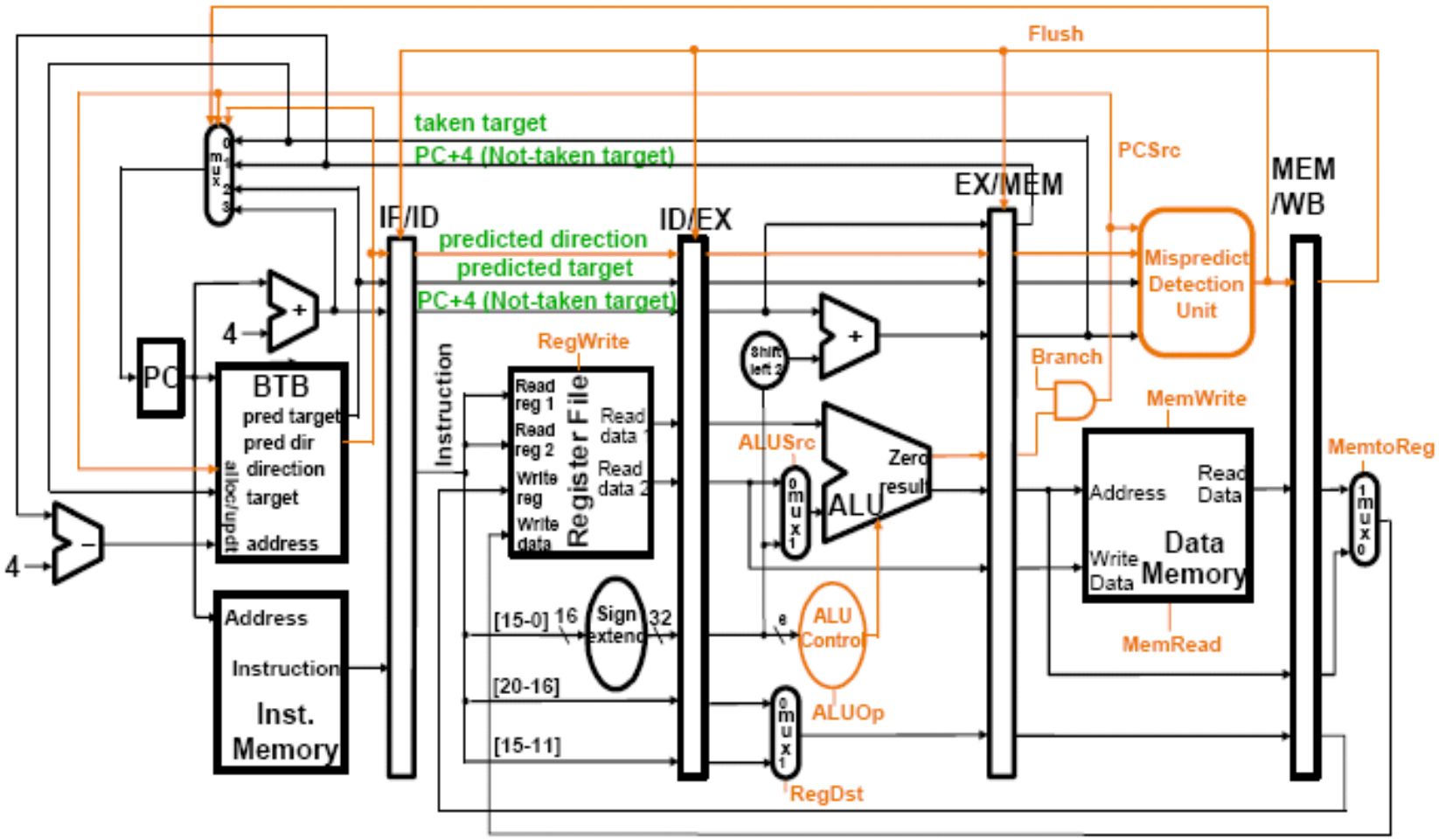
BTB

- **Allocation**
  - **Allocate instructions identified as branches (after decode)**
    - **Both conditional and unconditional branches are allocated**
  - **Not taken branches need not be allocated**
    - **BTB miss implicitly predicts not-taken**
- **Prediction**
  - **BTB lookup is done parallel to IC lookup**
  - **BTB provides**
    - **Indication that the instruction is a branch (BTB hits)**
    - **Branch predicted target**
    - **Branch predicted direction**
    - **Branch predicted type (e.g., conditional, unconditional)**
- **Update (when branch outcome is known)**
  - **Branch target**
  - **Branch history (taken / not-taken)**

BTB (cont)

- **Wrong prediction**
  - **Predict not-taken, actual taken**
  - **Predict taken, actual not-taken.**
- **In case of wrong prediction – flush the pipeline**
  - **Reset latches (same as making all instructions to be NOPs)**
  - **Select the PC source to be from the correct path**
    - **Need get the fall-through with the branch**
  - **Start fetching instruction from correct path**
- **Assuming P% correct prediction rate**
  - **20% of instructions are branches**
- **CPI new = 1 + (0.2 × (1-P)) × 3**
  - **For example, if P=0.7**
- **CPI new = 1 + (0.2 × 0.3) × 3 = 1.18**

Adding a BTB to the Pipeline

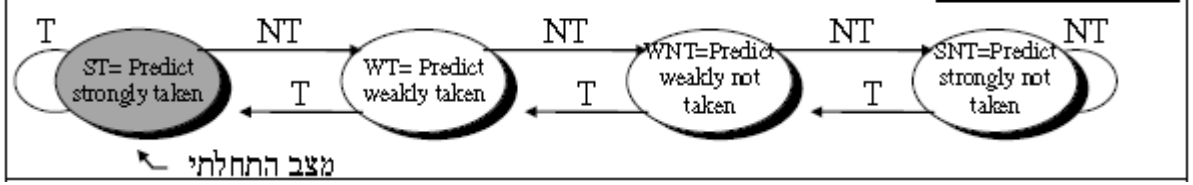


נתון מעבד , אשר משתמש במנגנון BTB לחיזוי התנהגות של פקודות branch.

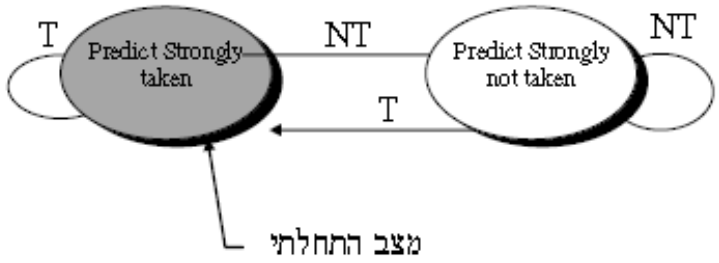
מריצים את קטע הקוד המייצג הבא על גבי המחשב:

hex כתובת הפקודה	הפקודה	משמעות
1000	Mov R1 #0	R1 ← 0
1004	Mov R2 #2	R2 ← 2
1008 *loop1*:	Inc R1	R1 ← R1+1
100C *loop2*:	Blt R1 R2 *loop1*	If (R1<R2) goto *loop1*
1010	Nop	Do Nothing
1014	Inc R2	R2 ← R2+1
1018	Nop	Do Nothing
101C	Blt R2 #4 *loop2*	If (R2<4) goto *loop2*
1020	Halt	End of program

מנגנון חיזוי ראשון: בשני סיביות – בדומה למכונת מצבים של Pentium



מנגנון חיזוי שני : חיזוי בסיבית אחת



להניח מצב התחלתי NT בדוגמא הנ"ל