תרגול 10

Multicycle architecture

אלון שקלר – אוניברסיטת תל אביב

למה Multicycle בקצרה

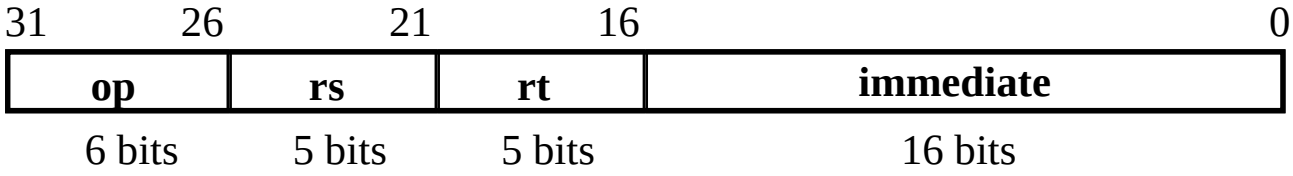
- ב-singlecycle הפקודה הארוכה ביותר קובעת את אורך המחזור וזה בזבזני.
- דורש שחלק מרכיבי הזיכרון יהיו כפולים
- ב-multicycle:
  - חיסכון בזמן: כל פקודה תקח את מספר היחידות השעון הנחוצות לה.
  - חיסכון ברכיבים: שימוש באותו רכיב בשלבים שונים של הפקודה.

תהליך מימוש פקודות - תזכורת

- ראשית, מתארים מה הפקודה עושה בשלבים:
  - מעבר נתונים (מקורות ויעד) תוך שימת דגש על האופרנדים
  - זיכרון/רגיסטרים
  - פעולות אריתמטיות שנעשות
- שנית, מממשים
  - מימוש מעבר הנתונים במערכת הקיימת:
    - האם הקווים הקיימים מספקים או
    - צריך להוסיף קווים נוספים
  - מימוש הבקרה במערכת הקיימת:
    - ***הוספת מצבים במכונת המצבים בהתאם לצורך***
    - קביעת ערכי הבקרה ביחידות הבקרה הקיימות
    - הוספת mux-ים במידת הצורך והגדרת קווי הבקרה שלהם
    - עדכון פונקציות קווי הבקרה הקיימים ע"י שילוב ערכי קווי הבקרה של הפקודה החדשה

הוספת הפקודה addi למחשב multicycle

- addi $t1, $t2, immediate



- תיאור:

1. הפקודה מובאת (ללא שינוי)
2. הפקודה מפוענחת ותוכן האוגרים נשלף (ללא שינוי)
3. הביצוע:

ALUOut = A + sign-extend(IR[15-0])

4. סיום הפקודה:

Reg[IR[20-16]] = ALUOut

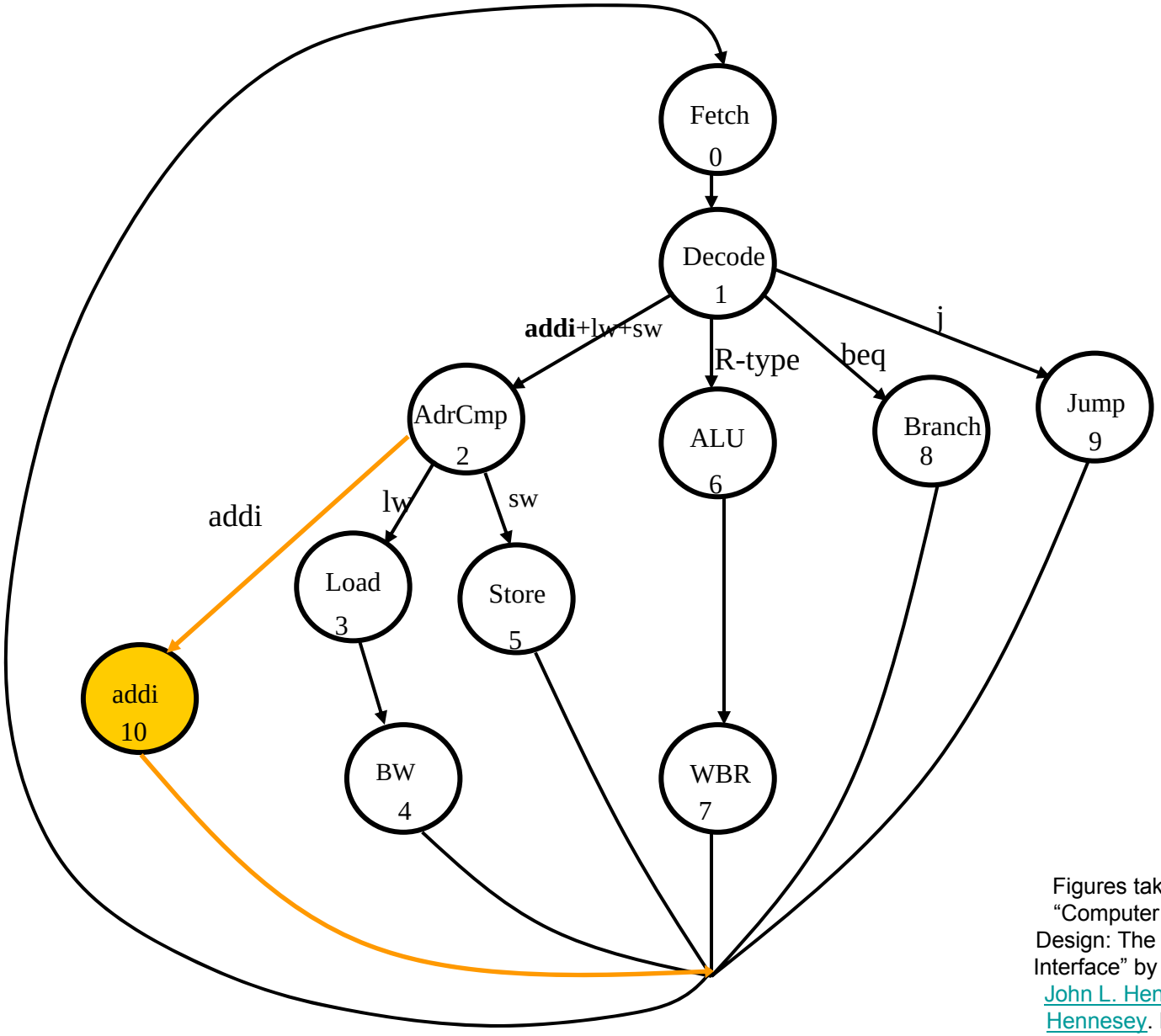
תזכורת

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR = Memory[PC]			
	PC = PC + 4			
Instruction	A = Reg [IR[25-21]]			
decode/register fetch	B = Reg [IR[20-16]]			
	ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
Execution, address	ALUOut = A op B	ALUOut = A + sign-extend	if (A ==B) then	PC = PC [31-28] II
computation, branch/		(IR[15-0])	PC = ALUOut	(IR[25-0]<<2)
jump completion				
Memory access or R-type	Reg [IR[15-11]] =	Load: MDR = Memory[ALUOut]		
completion	ALUOut	or		
		Store: Memory [ALUOut] = B		
Memory read completion		Load: Reg[IR[20-16]] = MDR		

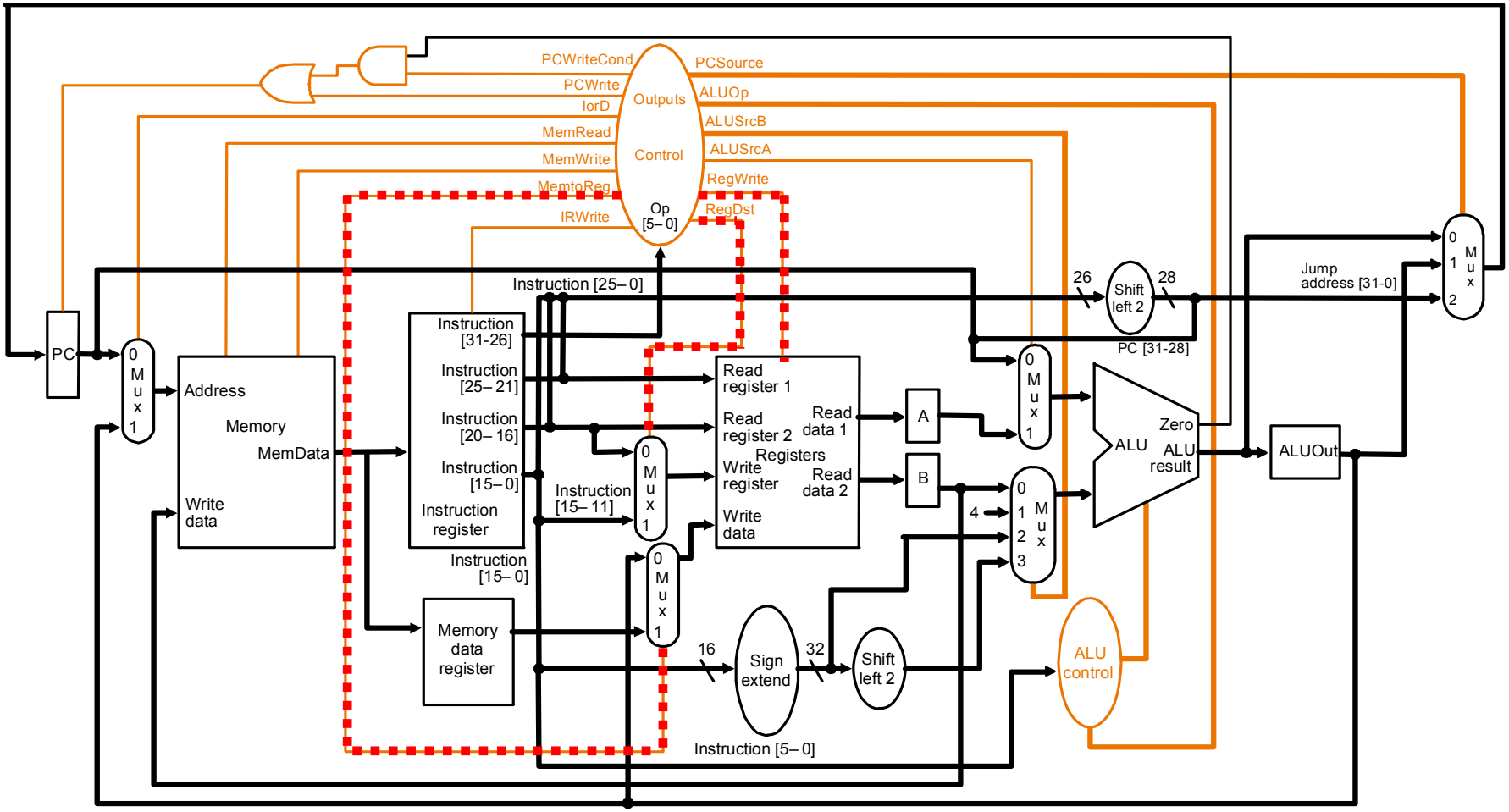
הוספת הפקודה addi למחשב multicycle

- קווי זרימת המידע מספיקים
- שלושת השלבים הראשונים זהים לאלה שמבוצעים ע"י פקודות הגישה לזיכרון (lw,sw) לכן נעשה בהם שימוש:
  - במכונת המצבים נוסיף addi על קשת המעבר בין מצב 1 ל-2.
- השלב הרביעי בביצוע הפקודה חדש ואינו מתאים למה שנתמך במכונת המצבים כרגע ולכן:
  - נוסיף מצב חדש (מספר 10) שיטפל בביצועו:
    - אליו נגיע ממצב 2 אם הפקודה היא addi.
    - ממנו נחזור למצב 0
- הביצוע ישלח ערכי בקרה מתאימים כדי שתוצאת ה-ALU היינו ALUOut תישמר במאגר האוגרים באוגר שמספרו מצוי במקומות 16-20 בפקודה.

הוספת הפקודה addi למחשב multicycle



Control signals for *addi*

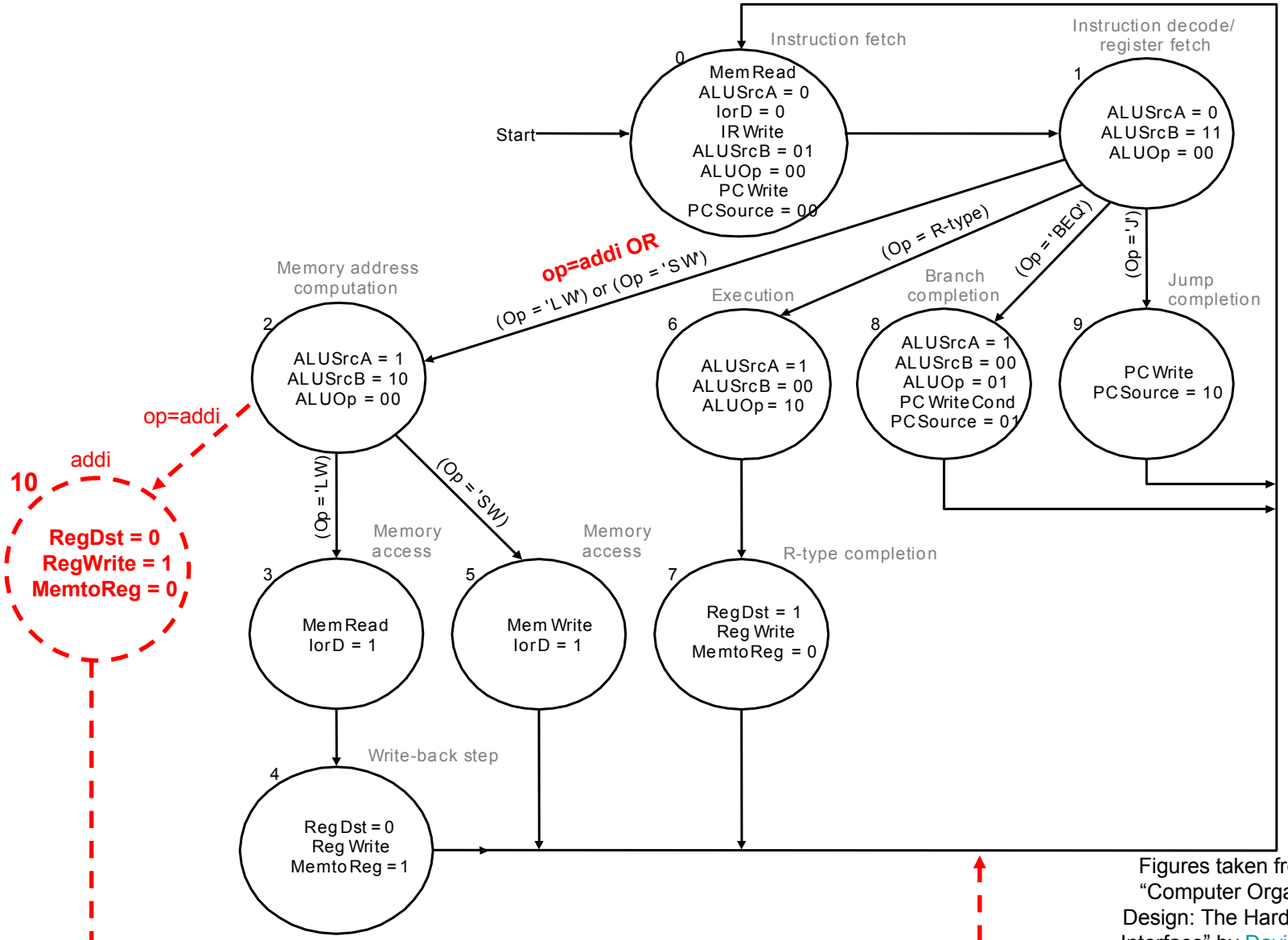


אלון שקלר – אוניברסיטת תל אביב

ערכי קווי הבקרה

- עבור המצבים הקיימים אין שינויים בערכי הבקרה משום שלא הורחב אף MUX קיים ולא נוספו MUX-ים חדשים.
- לכן נותר רק לציין את ערכי הבקרה עבור המצב החדש שהוספנו.
- במצב החדש נשמרת תוצאת החיבור שנתונה ב-ALUOut באוגר שמספרו נתון בסיביות 16-20
  - שמירה באוגר *rt* <- RegDst = 0, RegWrite=1
  - הערך שנשמר מגיע מ-ALUOut <- MemtoReg=0

הוספת הפקודה addi למחשב multicycle



הוספת הפקודה jal למחשב multicycle

- תיאור הביצוע:

1. הפקודה מובאת (ללא שינוי)
2. הפקודה מפוענחת ותוכן האוגרים נשלף (ללא שינוי)
3. הביצוע:

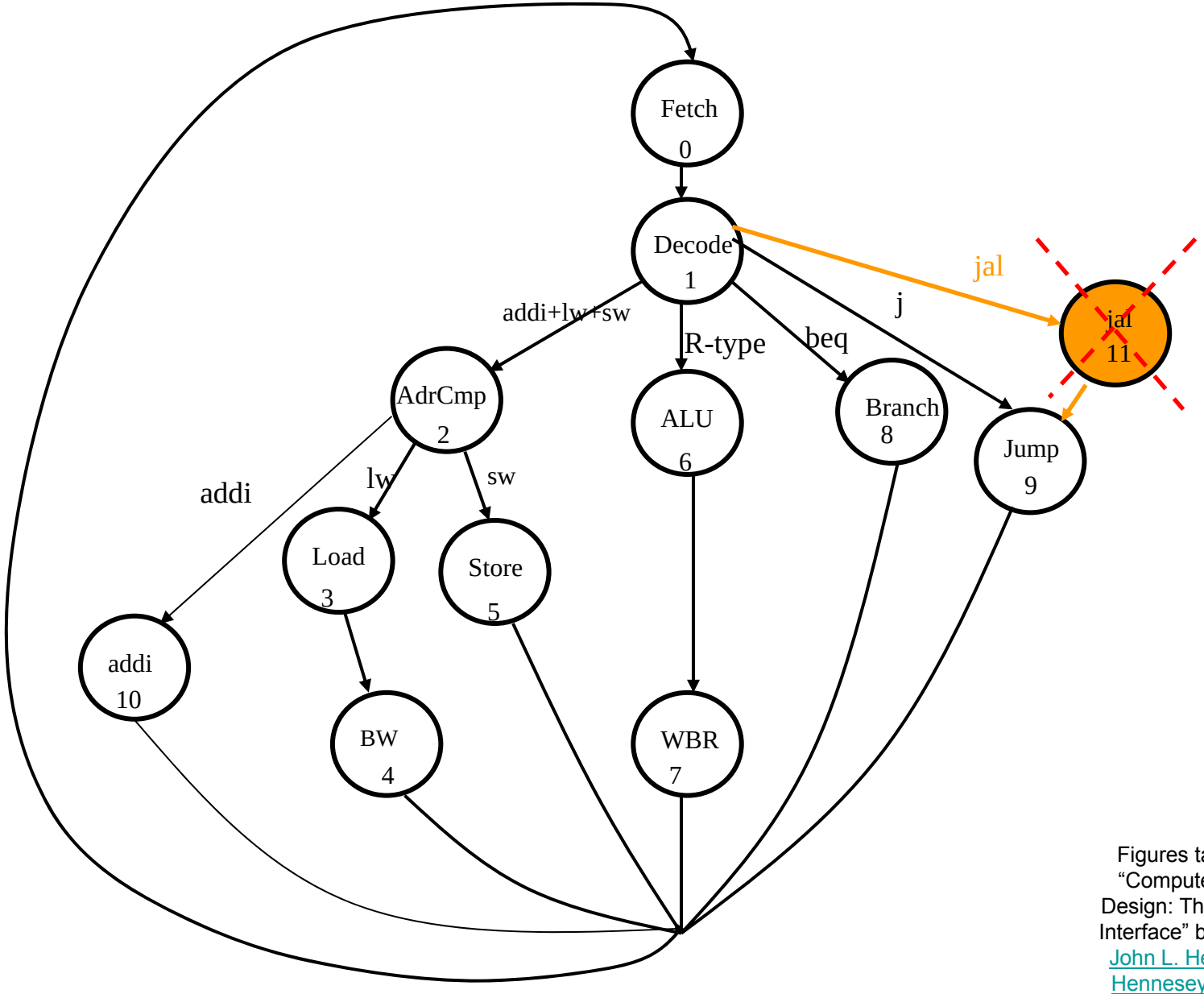
Reg[31] = PC; ← *link*

PC = PC[28-31] || (IR[0-25]<<2) ← כמו ב-jump

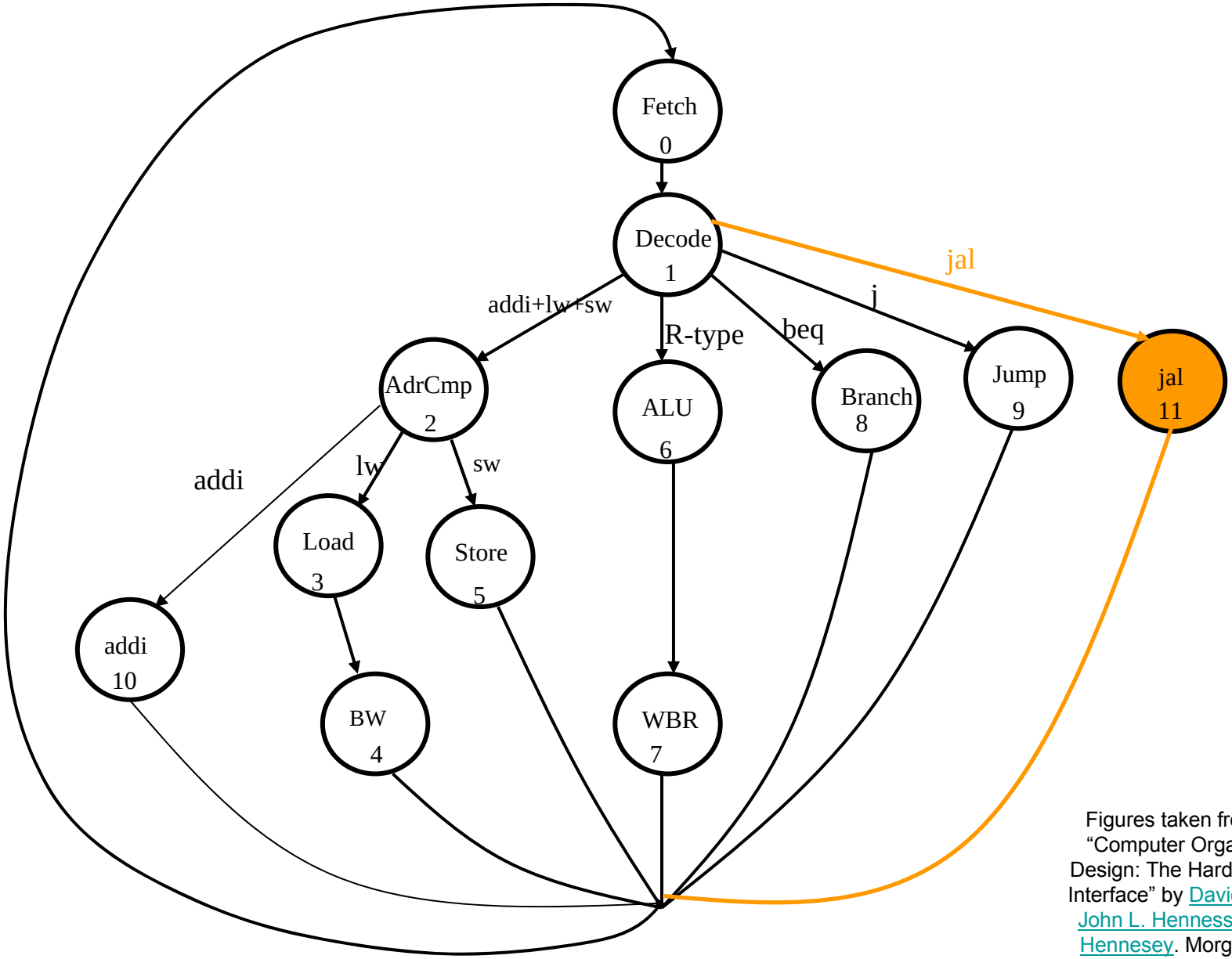
הוספת הפקודה jal למחשב multicycle

- קווי הנתונים (והבקרה) אינם מספיקים משום שיש להוסיף:
  - את PC כקלט ל-MemToReg mux
  - 31 כקלט ל- RegDst mux
- במכונת המצבים יש להוסיף מצב שמבצע בפועל את פעולת ה-jal ואז חוזר למצב 0

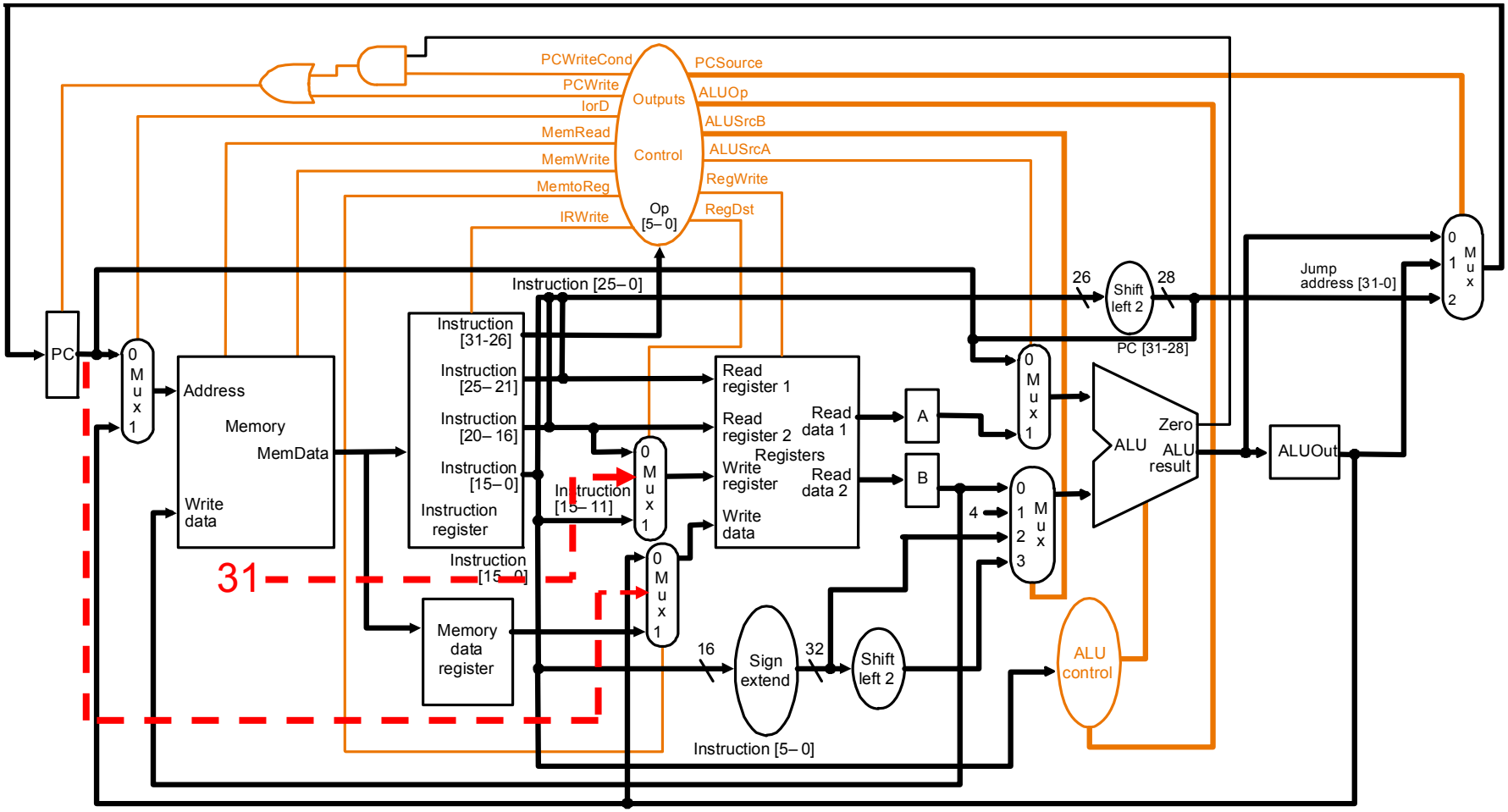
הוספת הפקודה jal למחשב multicycle



הוספת הפקודה jal למחשב multicycle



jal command – datapath additions



jal command – control path additions

- נשים לב שה-MUX-ים: MemtoReg, RegDst הורחבו ל-3 כניסות ולכן כניסות הבקרה שלהם תהיינה מורכבות מ-2 קווים.
- נגדיר את אפשרויות קווי הבקרה כך שלערכים המתאימים לערכים הישנים יהיה ב-MUX החדש 0 מוביל.
- ערך הבקרה החדש בשניהם הוא 10.

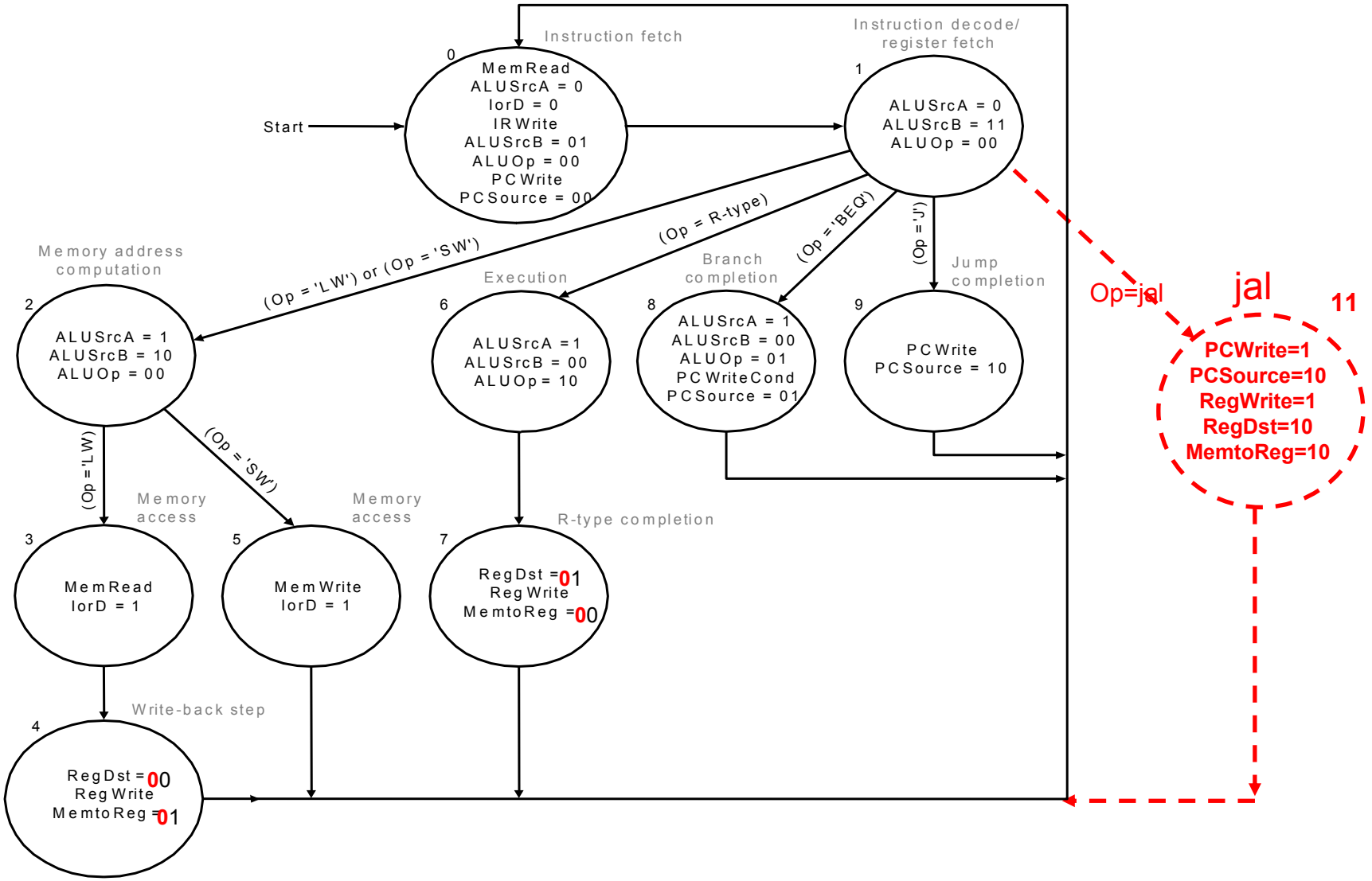
RegDst	
Control Code	Selected input
00	Bits 16-20
01	Bits 11-15
10	31

MemtoReg	
Control Code	Selected input
00	ALUOut
01	MDR
10	PC

jal command – control path additions

- יש לעדכן את ערכי הבקרה בכל המצבים בהתאם לערכי הבקרה החדשים.
- כמו כן יש להוסיף מצב חדש שמבצע את פעולת ה-jal:
  - כותב ל-PC את כתובת הקפיצה:
    PCWrite=1, PCSource=10
  - כותב ל-$ra את כתובת החזרה (PC):
    RegWrite=1, RegDst=10, MemtoReg=10

מכונת המצבים המעודכנת



דוגמא נוספת

- נגדיר את הפקודה wai (where am i):
  – הכנס את תוכן אוגר ה-PC של הפקודה (היינו תוכנו בעת ביצוע wai) אל האוגר שמצוין בשדה ה-*rt* בפקודה
- הנחות:
  – קווי מעבר הנתונים יישארו ללא שינוי
  – וכמובן, מחזור שעון אחד קצר מכדי לאפשר גם פעולה אריתמטית ב-ALU וגם גישה למאגר הרגיסטרים אם אחד תלוי בתוצאת האחר

הפקודה wai

מהלך הפקודה:

1. הפקודה מובאת (ללא שינוי)
2. הפקודה מפוענחת ותוכן האוגרים נשלף (ללא שינוי)
3. הביצוע:

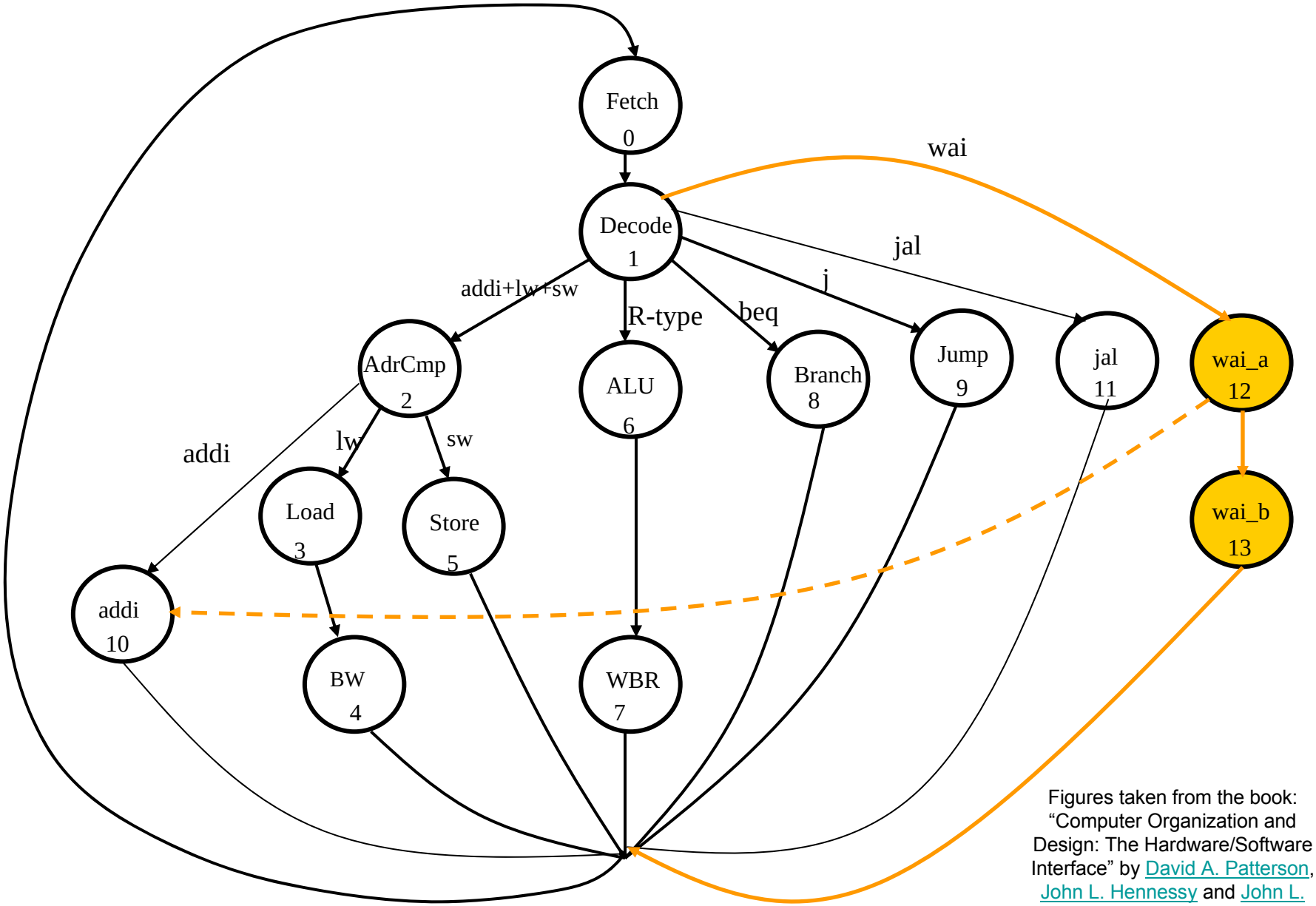
* היות ובשלב הראשון ה-PC מקודם ב-4 יש להחסיר ממנו 4 לפני הכנסתו לרגיסטר המבוקש:

a. ALUOut = PC – 4
b. Reg[IR[16-20]] = ALUOut

הפקודה wai

- מבחינת קווי הנתונים אין שינוי (לפי הנחה)
- מבחינת מכונת המצבים נוסיף שני מצבים חדשים:
  - 12 אחראי על חיסור 4 מ-PC והצבתו לתוך ALUOut
  - ו-13 שאחראי על הצבת ALUOut לרגיסטר המבוקש
- אל מצב 12 נגיע ממצב 1 במקרה והפקודה היא wai
- אל מצב 13 נגיע ממצב 12 ללא התניה
- ממצב 13 נחזור למצב 0

הוספת הפקודה wai למחשב multicycle



הפקודה wai – ערכי הבקרה

- מצב 12 אחראי על חיסור 4 מ-PC והצבתו לתוך ALUOut כלומר:
  - האופרנדים ל-ALU הם האוגר PC והקבוע 4:
    ALUSrcA=0, ALUSrcB=01
  - הפעולה שמתבצעת היא חיסור:
    ALUop=01
- מצב 13 אחראי על הצבת תוצאת החיסור לתוך האוגר המבוקש (סיביות 16-20):
  - RegDst=(0)0, MemtoReg=(0)0, RegWrite=1

בהנחה שפקודה זו היא מתווספת אחרי jal

הפקודה wai – מכונת המצבים