OOO Execution of Memory Operations

Reminders

- P6 L1 and L2 caches are non-Blocking
- RS solves false register dependencies issues
- MOB solves memory dependencies issues
  - Some memory dependencies can be resolved statically

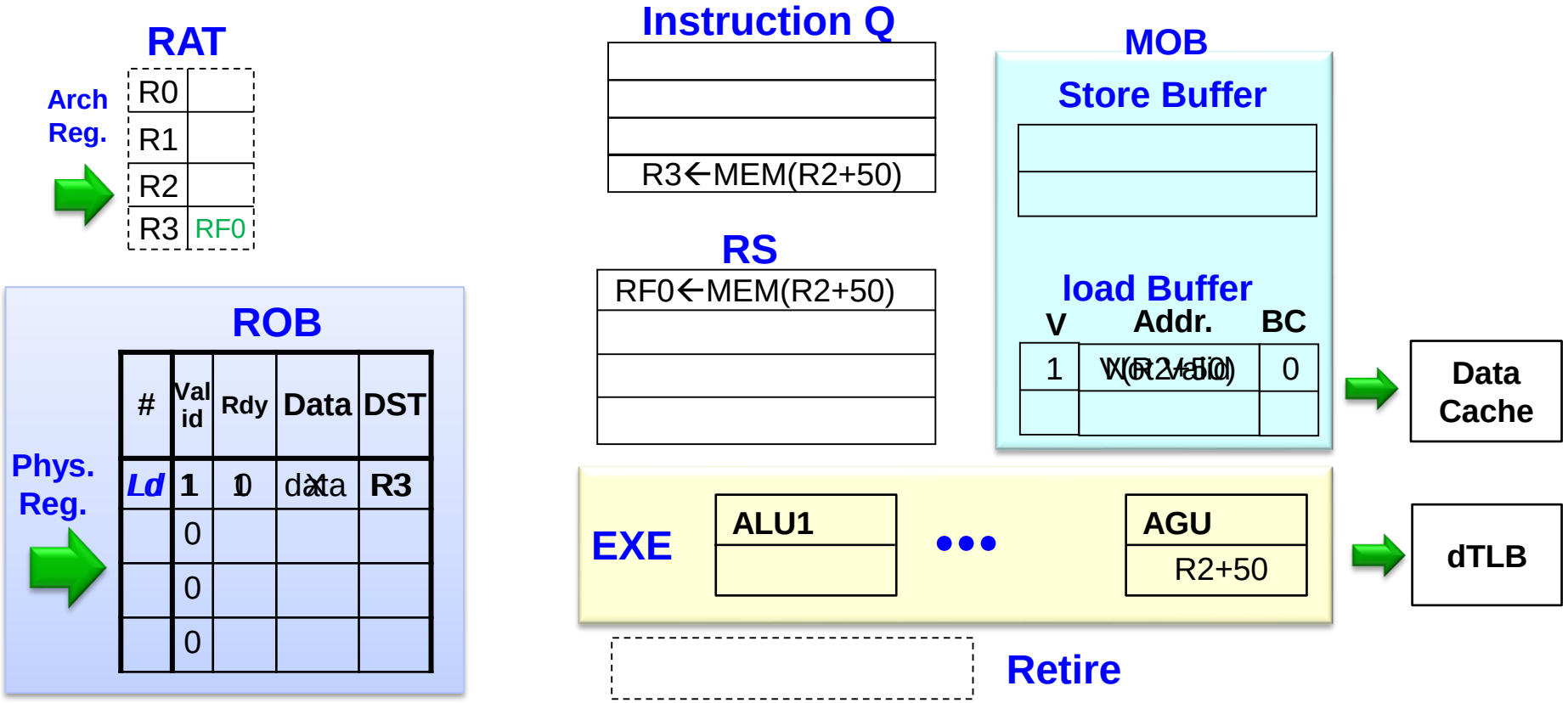
```
store r1,a
load r2,b
```

  - Problem: some cannot

```
store r1,[r3];
load r2,b
```

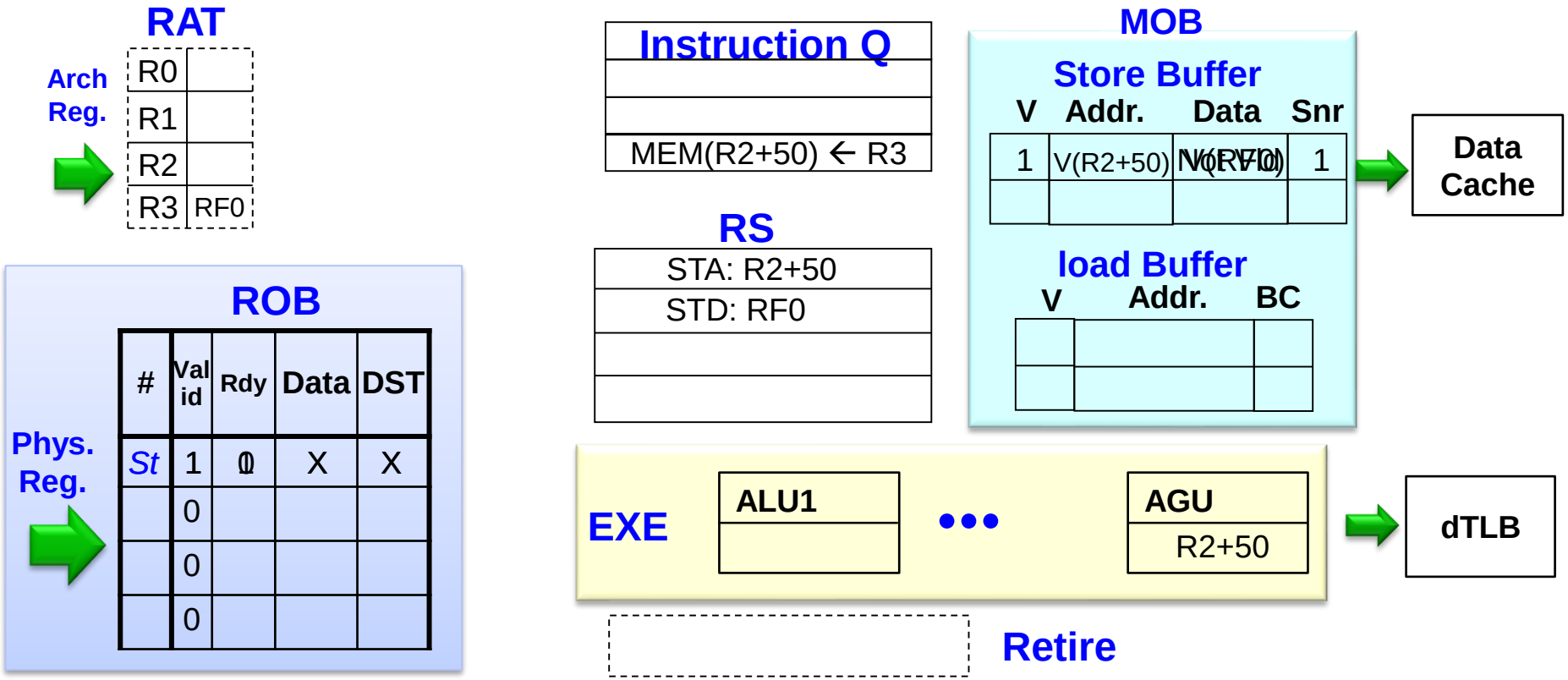
- Store/load are used a lot in X86 because of the small number of registers
- Stores are not executed out of order
  - Complicated to undo them
  - 2 uops: STA and STD
- Load are executed out of order
- Previous stores may have unknown (virtual) address

The life of a Load…



- 1 entry in the ROB, RS and Load Buffer + rename in RAT
- Dispatch Load address calculation to AGU when source is ready – Release RS entry
- AGU updates the address in the Load buffer. Pipeline proceeds to dTLB
- Load Buffer checks for blocking conditions and dispatches the Load to the DCU
- DCU sends the result to RS and updates the ROB with the load result
- Load will retire as any other instruction (when all previous instructions have retired) – RAT updated
- LB and ROB entry are released

The life of a Store…



- 1 entry in the ROB, 2 in the RS and 1 in the Store Buffer
- Dispatch Store address calculation to AGU when source is ready – Release RS entry
- AGU calculates the virtual address for STA → update the Store Buffer & provide address to depending loads.Store pipeline proceeds to dTLB → Physical address updated in the SB
- Dispatch STD (Store Data uop) when ready → update the Store Buffer & provide data to depending loads
- The Store Buffer updates the ROB entry with 'valid'
- The Store retires from the ROB ("in-order" as any other instruction = when all previous instructions retired)
- After retirement, Store is marked as **Senior Store** in the Store Buffer
- The Store buffer initiates a DCU write. When the write is done, the SB entry is freed

Question

- בשאלה זו נתייחס למעבד עם OOOE ו- Speculative Execution
- נתון קטע הקוד הבא:

```
1000    load    R2,R1,30        ; R2=m[R1+30]
1004    store   R2,20,R1        ; m[R2+20]=R1
1008    load    R3,R1,100       ; R3=m[R1+100]
100C    store   R1,40,R3        ; m[R1+40]=R3
1010    add     R1,R1,10        ; R1=R1+10
1014    blt     R1,100,1000     ; if (R1<100) PC=1000
```

- הנחות
  - בתחילת הביצוע בכל כתובת N בזיכרון קיים הערך, N וכן R1=R2=R3=10
  - למען פשטות נניח כי הכתובות בתוכנית הן פיזיות ואין צורך בתרגום.
  - המספרים בתוכנית ניתנים בבסיס 16.

המטמון במעבד

- L1 data cache, מחזיר data תוך מחזור שעון אחד, אך הוא ריק בתחילת הביצוע.
- L2 data cache מחזיר data תוך 7 מחזורי שעון, והוא מכיל את כל הכתובות המבוקשות כבר בתחילת הביצוע.
- גודל שורת מטמון הוא H80 בתים.
- מדיניות הכתיבה במטמונים היא no write allocate .

אלוקציה של פקודות

- בכל מחזור ניתן לבצע אלוקציה לארבע פקודות (ויש לפחות 4 פקודות מוכנות לאלוקציה)
- ה-ROB, MOB, וה- RS הם גדולים ואינם מתמלאים.

ביצוע של פקודות

- ישנן אינסוף יחידות ביצוע.
- פקודה יכולה להיכנס לביצוע במחזור שלאחר האלוקציה שלה בתנאי שכל הנתונים להם היא זקוקה כבר מוכנים.
- פקודה שממתינה לנתון יכולה להיכנס לביצוע מייד במחזור שלאחריו הוא מוכן.
- ביצוע פקודת ALU אורך מחזור שעון אחד.
- ביצוע פקודת branch אורך מחזור אחד.
  - אם החיזוי מתגלה כשגוי, במחזור הבא מבוצע flush (בזמן t+1).
  - הפקודות מהמסלול הנכון מבצעות אלוקציה 5 מחזורים לאחר flush (בזמן t+6).

ביצוע של פקודות – המשך

- פקודת load נשלחת לביצוע כאשר הנתונים לחישוב הכתובת מוכנים.
  - במחזור הראשון מחושבת הכתובת
  - במחזור השני מתורגמת הכתובת הוירטואלית לפיזית באמצעות ה-DTLB. כמו כן במחזור זה נבדקת תלות של ה-load ב-stores קודמים: עבור כל פקודת store הקודמת ל-load נבדק כי הכתובת של ה-store ידועה וכן מתקיים: או שהכתובת של ה-load שונה מהכתובת של ה-store, או ששתי הכתובות שוות, וה-data של ה-store כבר ידוע.
  - במחזור השלישי, במידה והבדיקה מצליחה, הנתון מתקבל מה-L1 cache (אם יש hit), או ישירות מה-MOB ע"י store to load forwarding
  - במידה והבדיקה מצליחה אך יש L1 cache miss וכן אין store to load forwarding, הנתון מתקבל במחזור העשירי מה- L2 cache.
  - במידה והבדיקה נכשלת, ה-load הוא חסום (blocked). כאשר מוסר תנאי החסימה, ה-load נשלח שוב לביצוע, ומדלגים על המחזור הראשון (מתחילים בבדיקת התנאי).
- פקודת store נשלחת לביצוע כאשר הנתונים לחישוב הכתובת מוכנים.
  - חישוב הכתובת אורך מחזור שעון אחד, ובסופו נכתבת הכתובת ל-MOB.
  - באופן בלתי תלוי, כאשר הנתון לכתיבה לזיכרון מוכן, במחזור הבא הוא נכתב ל-MOB

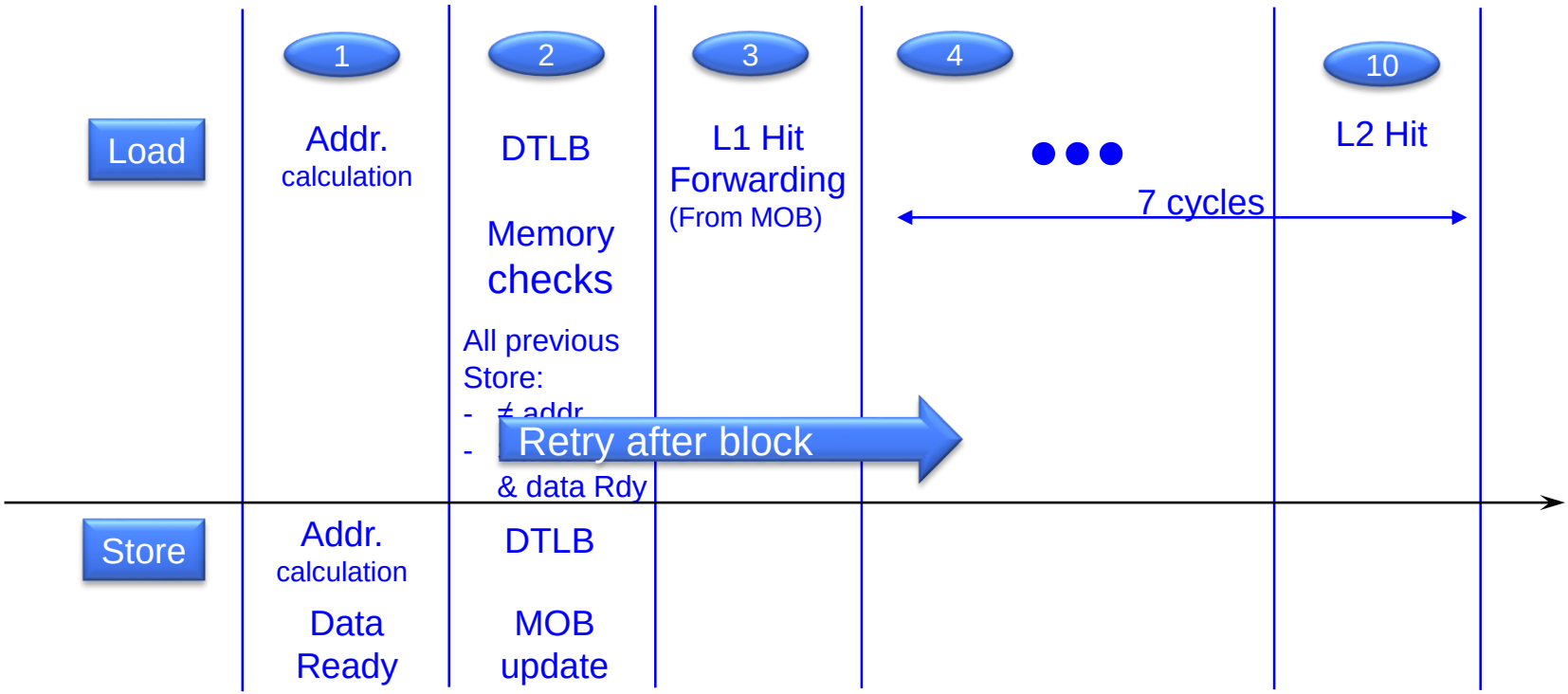
Commit של פקודות

- פקודה יכולה לבצע commit החל מהמחזור שלאחר סיום הביצוע שלה, ובתנאי שהפקודה שלפניה ביצעה/מבצעת commit. אין מגבלה על כמות הפקודות שמבצעות commit בכל מחזור.

- פקודת store מבצעת את הכתיבה אל ה-cache **לאחר** ה-commit.

Summary…

- 4 wide machine
- L1: 1 cycle L2: 7 cycles Alu, Branch: 1 cycle
- L1 empty / L2 always hits…
- Mispredict @ T:
  - T+1: Flush pipeline
  - T+6: Alloc on the good path



Arch. reg value after commit

Addr. for LD & ST

Data for LD & ST

Alloc Time 4 / cycle

Src reg:
Pi / Ri:
Store:
Src1: addr
Src2: data

Time Src ready

Time exe

0: ready
1: addr blocking
2: data not ready

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]															
1	store m[R2+20]=R1															
2	load R3=m[R1+100]															
3	store m[R1+40]=R3															
4	add R1=R1+10															
5	blt if (R1<100)															
6	load R2=m[R1+30]															
7	store m[R2+20]=R1															
8	load R3=m[R1+100]															
9	store m[R1+40]=R3															
10	add R1=R1+10															
11	blt if (R1<100)															

Fill this table…

הנחיות למילוי הטבלה

- R1, R2, R3 - ערכי הרגיסטרים הארכיטקטוניים לאחר commit.
  יש להקיף בעיגול את הערך של הרגיסטר הארכיטקטוני שאליו הפקודה כותבת.
  במידה והפקודה אינה מגיעה ל-commit יש להשאיר שדות אלה ריקים.
- addr– כתובת הגישה לזיכרון – עבור פקודות load ו-store בלבד.
- data– ערך זיכרון שנקרא או נכתב – עבור פקודות load ו-store בלבד.
- T(alloc): הזמן בו מבוצעת אלוקציה לפקודה (ארבע פקודות בכל מחזור, החל מ- T=1)
- src1, src2: מספרי הרגיסטרים המשמשים כ-sources לפקודה:
  - **Pi** עבור רגיסטר פיזי (מס' הרגיסטר הפיזי של פקודה הוא מס' השורה שלה בטבלה)
  - **Ri** במידה וקוראים ישירות את הרגיסטר הארכיטקטוני.
- עבור store: src1– הרגיסטר המשמש לחישוב הכתובת. src2– הרגיסטר המכיל את הנתון.
- Imm– במידה ולפקודה יש Imm, ערך ה- Imm.
- T(src1 ready) , T(src2 ready): הזמן בו מוכן כל אחד ערכי ה-sources לפקודה.
  אם ה-src מוכן בזמן האלוקציה, אז זמן זה יהיה שווה לזמן האלוקציה.
  אם הפקודה שמחשבת את הערך של src מסיימת ביצוע בזמן T, ה-src מוכן בזמן T.

הנחיות למילוי הטבלה – המשך

- T(exe): הזמן בו הפקודה נשלחת לביצוע.
  אם כל ה-src-ים של פקודה מוכנים בזמן T, ניתן לשלוח את הפקודה לביצוע בזמן T+1.
- Load block code (רלוונטי רק בפקודות load): קוד החסימה של ה-load.
  0 – אין חסימה.
  1 – חסימה כתוצאה מ- unknown store address
  2 – חסימה כתוצאה מ- waiting for store data
- במידה וה-load נחסם יותר מפעם אחת, יש לרשום את כל קודי החסימה.
- T(data ready):
  - עבור store: הזמן בו ה-data לכתיבה לזיכרון וגם הכתובת מוכנים.
  - עבור load: הזמן בו מתקבל ה-data (מה-cache או ישירות מה-MOB).
- T(commit): הזמן בו הפקודה מבצעת commit

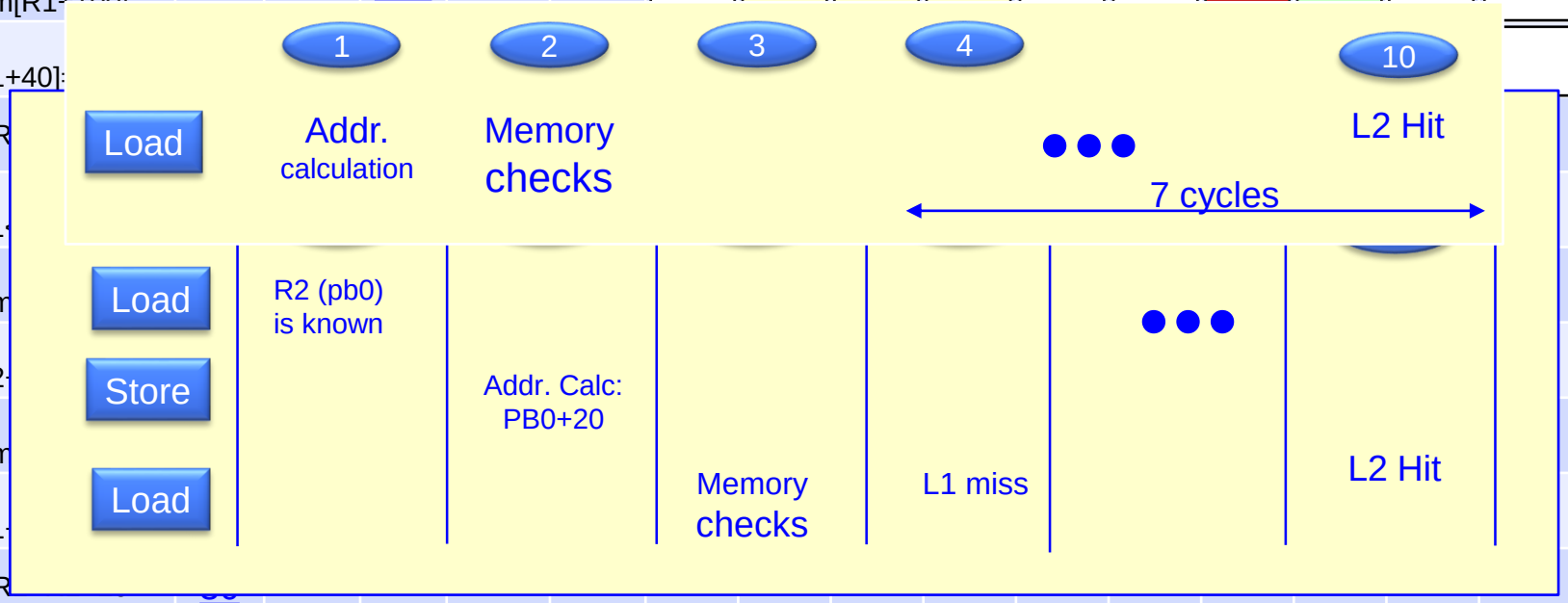
מילוי הטבלה – שלב א'

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40										
1	store m[R2+20]=R1				60	10										
2	load R3=m[R1+100]			110	110	110										
3	store m[R1+40]=R3				50	110										
4	add R1=R1+10	20														
5	blt if (R1<100)	20	40	110												
6	load R2=m[R1+30]		110		50	110										
7	store m[R2+20]=R1				130	20										
8	load R3=m[R1+100]			120	120	120										
9	store m[R1+40]=R3				60	120										
10	add R1=R1+10	30														
11	blt if (R1<100)	30	110	120												

Src reg:
Pi / Ri:
Store:
Src1: addr
Src2: data

0: ready
1: addr blocking
2: data not ready

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40										
1	store m[R2+20]=R1				60	10	.	.			.					
2	load R3=m[R1+100]			110	110	110	.							.		
3	store m[R1+40]=															
4	add R															
5	blt if (R1															
6	load R2=m															
7	store m[R2															
8	load R3=m															
9	store m[R1															
10	add R															
11	blt if (R1<100)	30	110	120												



Src reg:
Pi / Ri:
Store:
Src1: addr
Src2: data

0: ready
1: addr blocking
2: data not ready

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13
2	load R3=m[R1+100]			110	110	110	1	R1		100	1		2	1	21	22
3	store m[R1+40]=R3				50	110	1	R1	P2	40	1	21	Std:22 Sta: 2		22	23
4	add R1=R1+10	20														
5	blt if (R1<100)	20	40	110												
6	load R2=m[R1+30]		110		50	110										
7	store m[R2+20]=R1				130	20										
8	load R3=m[R1+100]			120	120	120										
9	store m[R1+40]=R3				60	120										
10	add R1=R1+10	30														
11	blt if (R1<100)	30	110	120												

Src reg:
Pi / Ri:
Store:
Src1: addr
Src2: data

0: ready
1: addr blocking
2: data not ready

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13
2	load R3=m[R1+100]			110	110	110	1	R1		100	1		2	1	21	22
3	store m[R1+40]=R3				50	110	1	R1	P2	40	1	21	Std:22 Sta: 2		22	23
4	add R1=R1+10	20					2	R1		10	2		3			23
5	blt if (R1<100)	20	40	110			2	P4		100	3		4			23
6	load R2=m[R1+30]		110		50	110	2	P4		30	3		4	1, 2	24	25
7	store m[R2+20]=R1				130	20	2	P6	P4	20	24	3	Std:4 Sta:25		25	26
8	load R3=m[R1+100]			120	120	120										
9	store m[R1+40]=R3				60	120										
10	add R1=R1+10	30														
11	blt if (R1<100)	30	110	120												

Src reg:
Pi / Ri:
Store:
Src1: addr
Src2: data

0: ready
1: addr blocking
2: data not ready

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13
2	load R3=m[R1+100]			110	110	110	1	R1		100	1		2	1	21	22
3	store m[R1+40]=R3				50	110	1	R1	P2	40	1	21	Std:22 Sta: 2		22	23
4	add R1=R1+10	20					2	R1		10	2		3			23
5	blt if (R1<100)	20	40	110			2	P4		100	3		4			23
6	load R2=m[R1+30]		110		50	110	2	P4		30	3		4	1, 2	24	25
7	store m[R2+20]=R1				130	20	2	P6	P4	20	24	3	Std:4 Sta:25		25	26
8	load R3=m[R1+100]			120	120	120	3	P4		100	3		4	1	27	28
9	store m[R1+40]=R3				60	120	3	P8	P4	40	27	3	Sta:4 Std:28		28	29
10	add R1=R1+10	30					3	P4		10	3		4			29
11	blt if (R1<100)	30	110	120			3	P10		100	4		5			29

Question 2
Top-Down Analysis

Question 2

- RS: 6 entries
  - RS entry is freed when the instruction is dispatched to execution
  - If uop cannot allocate (due to RS full)
    - Stall until free entry
- Stores occupy 2 entries in the RS
  - STA + STD
  - Store instructions are always allocated in the same cycle
- The first branch mispredicts
  - Instruction from the wrong path are flushed from RS at T(branch exe)+1
  - Instructions from the correct path can be allocated at T(branch exe)+1
- The instruction queue is always full until the second branch. After it it's empty.

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit	#RS Entries
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12	
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13	
2	load R3=m[R1+100]			110	110	110	1	R1		100	1		2	1	21	22	
3	store m[R1+40]=R3				50	110	1	R1	P2	40	1	21	Std:22 Sta: 2		22	23	
4	add R1=R1+10	20					2	R1		10	2		3			23	
5	blt if (R1<100)	20	40	110			2	P4		100	3		4			23	
6	load R2=m[R1+30]		110		50	110	2	P4		30	3		4	1, 2	24	✗	
7	store m[R2+20]=R1				130	20			P4	20	24	3	Sta:4 Std:25		25	✗	
8	load R3=m[R1+100]			120	120	120				100	4			1	27	✗	
9	store m[R1+40]=R3				60	120	4	P8	P4	40	27	4	Sta:5 Std:28		28	✗	
10	add R1=R1+10	30								10	3		4			23	
11	blt if (R1<100)	30	110	120			5	P10		100	4		5			23	

@ Cycle 2

@ Cycle 3

@ Cycle 4

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T com mit	#RS Entries
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12	0+1=1
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13	+2=3
2	load R3=m[R1+100]			110	110	110	1	R1		100	1		2	1	21	22	+1=4
3	store m[R1+40]=R3				50	110	1	R1	P2	40	1	21	Std:22 Sta: 2		22	23	+2=6
4	add R1=R1+10	20					2	R1		10	2		3			23	6-4 +1=3
5	blt if (R1<100)	20	40	110			2	P4		100	3		4			23	+1=4
6	load R2=m[R1+30]		110		50	110	2	P4		30	3		4	1, 2	24	✗	+1=5
7	store m[R2+20]=R1				130	20	3	P6	P4	20	24	3	Std:4 Sta:25		25	✗	5-1 +2=6
8	load R3=m[R1+100]			120	120	120	4	P4		100	4		5	1	27	✗	6-3 +1=4
9	store m[R1+40]=R3				60	120	4	P8	P4	40	27	4	Sta:5 Std:28		28	✗	+2=6
10	add R1=R1+10	30					5	P4		10	3		4			23	2+1=3
11	blt if (R1<100)	30	110	120			5	P10		100	4		5			23	+1=4

RS full @ Cycle 2

RS full @ Cycle 3

RS full @ Cycle 4

Fill the top-down table

Cycle	1	2	3	4	5	6	7	8	9	10	11
Back-end Stall	0										
Alloc Slot 0	✓										
Alloc Slot 1	✓										
Alloc Slot 2	✓										
Alloc Slot 3	✓										
Frontend Bound											
Backend Bound											
Retiring	4										
Bad Speculation											

Bad Speculation	Front end bound	Backend bound	Retiring

Fill the top-down table

Cycle	1	2	3	4	5	6	7	8	9	10	11
Back-end Stall	0	1	1	1	1	0	0	0	0		
Alloc Slot 0	✓	✓	✓	✓	✓						
Alloc Slot 1	✓	✓		✓	✓						
Alloc Slot 2	✓	✓									
Alloc Slot 3	✓										
Frontend Bound					2						
Backend Bound		1	3	2							
Retiring	4	2			2						
Bad Speculation		1	1	2							

Bad Speculation	Front end bound	Backend bound	Retiring
4 / 20 = 20%	2 / 20 = 10%	6 / 20 = 30%	8 / 20 = 40%

Backup

Question

- בשאלה זו נתייחס למעבד עם OOOE ו- Speculative Execution
- נתון קטע הקוד הבא:

```
1000    load    R2,R1,30        ; R2=m[R1+30]
1004    store   R2,20,R1        ; m[R2+20]=R1
1008    load    R3,R1,100       ; R3=m[R1+100]
100C    store   R1,40,R3        ; m[R1+40]=R3
1010    add     R1,R1,10        ; R1=R1+10
1014    blt     R1,100,1000     ; if (R1<100) PC=1000
```

- הנחות
  - בתחילת הביצוע בכל כתובת N בזיכרון קיים הערך, N וכן R1=R2=R3=10
  - למען פשטות נניח כי הכתובות בתוכנית הן פיזיות ואין צורך בתרגום.
  - המספרים בתוכנית ניתנים בבסיס 16.

המטמון במעבד

- L1 data cache, מחזיר data תוך מחזור שעון אחד, אך הוא ריק בתחילת הביצוע.
- L2 data cache מחזיר data תוך 7 מחזורי שעון, והוא מכיל את כל הכתובות המבוקשות כבר בתחילת הביצוע.
- גודל שורת מטמון הוא H80 בתים.
- מדיניות הכתיבה במטמונים היא write no-allocate .

אלוקציה של פקודות

- בכל מחזור ניתן לבצע אלוקציה לארבע פקודות (ותמיד יש לפחות 4 פקודות מוכנות לאלוקציה)
- ה-ROB, MOB, וה- RS הם גדולים ואינם מתמלאים.

ביצוע של פקודות

- ישנן אינסוף יחידות ביצוע.
- פקודה יכולה להיכנס לביצוע במחזור שלאחר האלוקציה שלה בתנאי שכל הנתונים להם היא זקוקה כבר מוכנים.
- פקודה שממתינה לנתון יכולה להיכנס לביצוע מייד במחזור שלאחריו הוא מוכן.
- ביצוע פקודת ALU אורך מחזור שעון אחד.
- ביצוע פקודת branch אורך מחזור אחד.
  - פקודת הקפיצה נחזית כ-Taken (והחיזוי נכון)

```
1000    load    R2,R1,30    ; R2=m[R1+30]
1004    store   R2,20,R1    ; m[R2+20]=R1
1008    load    R3,R1,100   ; R3=m[R1+100]
100C    store   R1,40,R3    ; m[R1+40]=R3
1010    add     R1,R1,10    ; R1=R1+10
1014    blt     R1,100,1000         ; if (R1<100) PC=1000
```

ביצוע של פקודות – המשך

- פקודת load נשלחת לביצוע כאשר הנתונים לחישוב הכתובת מוכנים.
  - במחזור הראשון מחושבת הכתובת
  - במחזור השני מתורגמת הכתובת הוירטואלית לפיזית באמצעות ה-DTLB. כמו כן במחזור זה נבדקת תלות של ה-load ב-stores קודמים: עבור כל פקודת store הקודמת ל-load נבדק כי הכתובת של ה-store ידועה וכן מתקיים: או שהכתובת של ה-load שונה מהכתובת של ה-store, או ששתי הכתובות שוות, וה-data של ה-store כבר ידוע.
  - במחזור השלישי, במידה והבדיקה מצליחה, הנתון מתקבל מה-L1 cache (אם יש hit), או ישירות מה-MOB ע"י store to load forwarding
  - במידה והבדיקה מצליחה אך יש L1 cache miss וכן אין store to load forwarding, הנתון מתקבל במחזור העשירי מה- L2 cache.
  - במידה והבדיקה נכשלת, ה-load הוא חסום (blocked). כאשר מוסר תנאי החסימה, ה-load נשלח שוב לביצוע, ומדלגים על המחזור הראשון (מתחילים בבדיקת התנאי).
- פקודת store מורכבת מ-STA ו-STD
  - חישוב הכתובת (STA) אורך מחזור שעון אחד, ובסופו נכתבת הכתובת ל-MOB
  - באופן בלתי תלוי, כאשר הנתון לכתיבה לזיכרון מוכן (STD), במחזור הבא הוא נכתב ל-MOB

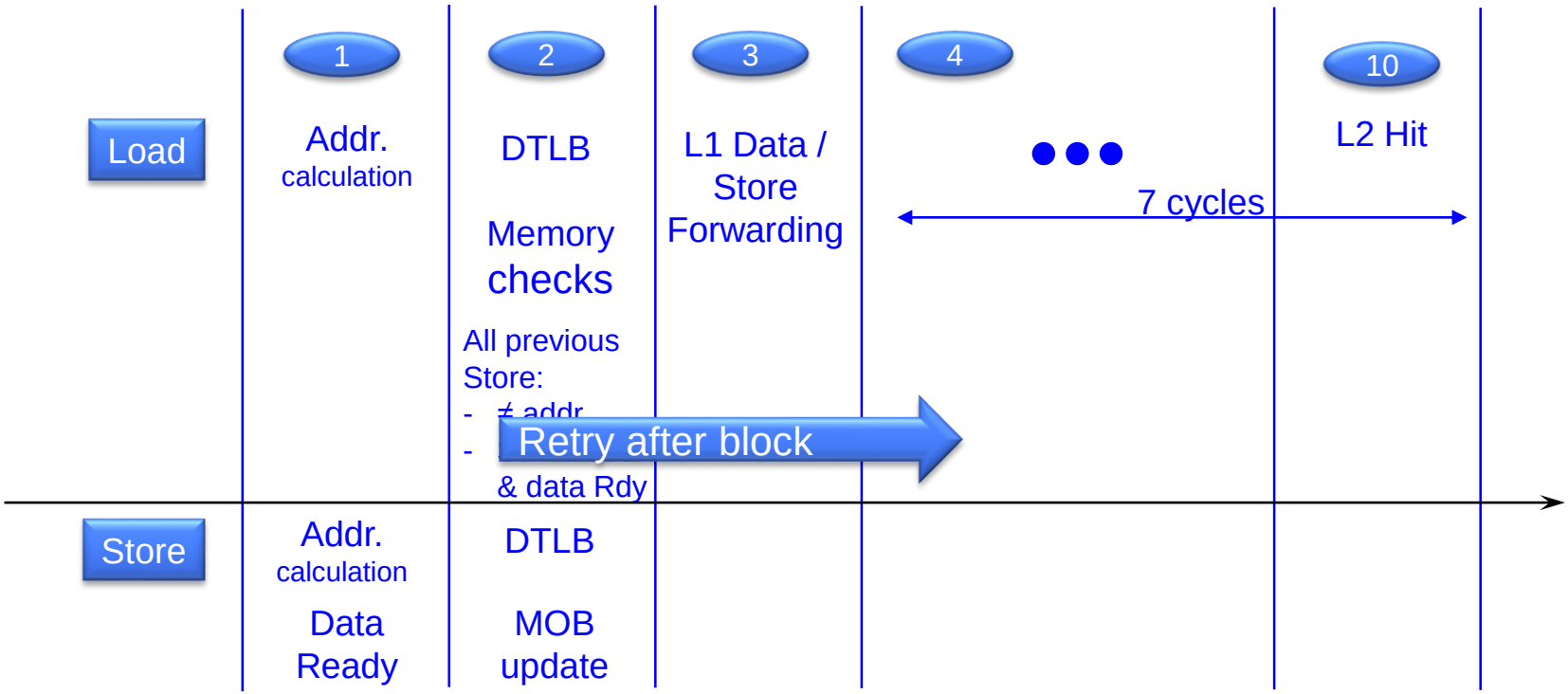
Commit של פקודות

- פקודה יכולה לבצע commit החל מהמחזור שלאחר סיום הביצוע שלה, ובתנאי שהפקודה שלפניה ביצעה/מבצעת commit. אין מגבלה על כמות הפקודות שמבצעות commit בכל מחזור.

- פקודת store מבצעת את הכתיבה אל ה-cache **לאחר** ה-commit.

Summary…

- 4 wide machine
- L1: 1 cycle L2: 7 cycles Alu, Branch: 1 cycle
- L1 empty / L2 always hits…
- Mispredict @ T:
  - T+1: Flush pipeline
  - T+6: Alloc on the good path



Arch. reg value after commit

Addr. for LD & ST

Data for LD & ST

Alloc Time 4 / cycle

Src reg:
Pi / Ri:
Store:
Src1: addr
Src2: data

Time Src ready

Time exe

0: ready
1: addr blocking
2: data not ready

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]															
1	store m[R2+20]=R1															
2	load R3=m[R1+100]															
3	store m[R1+40]=R3															
4	add R1=R1+10															
5	blt if (R1<100)															
6	load R2=m[R1+30]															
7	store m[R2+20]=R1															
8	load R3=m[R1+100]															
9	store m[R1+40]=R3															
10	add R1=R1+10															
11	blt if (R1<100)															

Fill this table…

הנחיות למילוי הטבלה

- R1, R2, R3 - ערכי הרגיסטרים הארכיטקטוניים לאחר commit.
  יש להקיף בעיגול את הערך של הרגיסטר הארכיטקטוני שאליו הפקודה כותבת.
  במידה והפקודה אינה מגיעה ל-commit יש להשאיר שדות אלה ריקים.
- addr– כתובת הגישה לזיכרון – עבור פקודות load ו-store בלבד.
- data– ערך זיכרון שנקרא או נכתב – עבור פקודות load ו-store בלבד.
- T(alloc): הזמן בו מבוצעת אלוקציה לפקודה (ארבע פקודות בכל מחזור, החל מ- T=1)
- src1, src2: מספרי הרגיסטרים המשמשים כ-sources לפקודה:
  - **Pi** עבור רגיסטר פיזי (מס' הרגיסטר הפיזי של פקודה הוא מס' השורה שלה בטבלה)
  - **Ri** במידה וקוראים ישירות את הרגיסטר הארכיטקטוני.
- עבור store: src1– הרגיסטר המשמש לחישוב הכתובת (פרמטר יחיד של STA).
  src2– הרגיסטר המכיל את הנתון (פרמטר יחיד של STD).
- Imm– במידה ולפקודה יש Immediate - ערכו.
- T(src1 ready) , T(src2 ready): הזמן בו מוכן כל אחד מערכי ה-sources.
  אם ה-src מוכן בזמן האלוקציה, אז זמן זה יהיה שווה לזמן האלוקציה.
  אם הפקודה שמחשבת את הערך של src מסיימת ביצוע בזמן T, ה-src מוכן בזמן T.

הנחיות למילוי הטבלה – המשך

- T(exe): הזמן בו הפקודה נשלחת לביצוע.
  אם כל ה-src-ים של פקודה מוכנים בזמן T, ניתן לשלוח את הפקודה לביצוע בזמן T+1.
- Load block code (רלוונטי רק בפקודות load): קוד החסימה של ה-load.
  0 – אין חסימה.
  1 – חסימה כתוצאה מ- unresolved store address
  2 – חסימה כתוצאה מ- waiting for store data
- במידה וה-load נחסם יותר מפעם אחת, יש לרשום את כל קודי החסימה.
- T(data ready):
  - עבור store: הזמן בו ה-data לכתיבה לזיכרון וגם הכתובת מחושבים.
  - עבור load: הזמן בו מתקבל ה-data (מה-cache או ישירות מה-MOB).
- T(commit): הזמן בו הפקודה מבצעת commit

שלב א' – מילוי השדות הנוגעים לערכים

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40										
1	store m[R2+20]=R1				60	10										
2	load R3=m[R1+100]			110	110	110										
3	store m[R1+40]=R3				50	110										
4	add R1=R1+10	20														
5	blt if (R1<100)	20	40	110												
6	load R2=m[R1+30]		110		50	110										
7	store m[R2+20]=R1				130	20										
8	load R3=m[R1+100]			120	120	120										
9	store m[R1+40]=R3				60	120										
10	add R1=R1+10	30														
11	blt if (R1<100)	30	110	120												

פקודה 0: load R2=m[R1+30]

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12

- T alloc: אלוקציה של פקודה 0: מתחילים לעשות אלוקציה כבר במחזור הראשון, כלומר מחזור מס' 1 (שם מביאים בסה"כ ארבע פקודות).
- src1: המקור היחיד בפקודה זו הוא R1.
- imm: הערך המידי שלה הוא 30.
- T src1 ready: מתי המקור הראשון מוכן? מכיוון שזו הפקודה הראשונה היעד הוא רגיסטר ארכ' ולא מחכים לאף פקודה אחרת בשבילו, הוא מוכן מיד, כלומר במחזור אחרי האלוקציה: 1.
- T exe: הפקודה נשלחת לביצוע במחזור אחרי שהמקורות מוכנים - מחזור 2 .
- Load block code: אין לפניה שום store ולכן אין חסימה - כותבים 0 .
- T data ready: מתי יגיע אליי ה-data? בפקודת load ניגשים אל המטמון. ב-L1 יהיה לנו Miss (כי הוא ריק בתחילת הריצה) ונתון שב-L2 תמיד יש hit - חשוב מאוד להבין בדיוק באיזה מחזור יגיע המידע: מחשבים את הכתובת במחזור 2, במחזור 3 רואים שאין תנאי חסימה, במחזור 4 מקבלים החטאה במטמון L1 וממחזור 5 (כולל) לוקח שבעה מחזורים להביא את המידע - לכן הוא מגיע במחזור 11 (ready)
- T commit: הפקודה עושה Retire במחזור אחרי הגעת המידע שלה, כלומר במחזור ה-12.

פקודה 0: load R2=m[R1+30]

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12

פקודה 1: store m[R2+20]=R1 (שקף 1/2)

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13

הפקודה הבאה היא פקודת store ויש לה שני מקורות - הראשון (src1) משמש לחישוב הכתובת והשני (src2) לחישוב המידע.

- האלוקציה נעשית במחזור 1 - אותו זמן של הקצאת הפקודה הראשונה כי עושים אלוקציה לארבע פקודות בכל מחזור.
- המקור הראשון הוא הרגיסטר הפיזי שאליו ממופה r2 ע"י הפקודה הקודמת - בשאלה מניחים שכל פקודה שומרת ערך אל הרגיסטר הפיזי שמספרו כמס' השורה שלה - כך זה גם נעשה ב-ROB אמתי. כלומר הפקודה בשורה 1 מחכה לרגיסטר הפיזי P0 (זה שהפקודה בשורה 0 כותבת אליו את התוצאה שלה).
- המקור השני הוא R1, בשביל חישוב ה-data שנכתב.

פקודה 1: store m[R2+20]=R1 (שקף 2/2)

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13

- המקור הראשון מוכן במחזור לאחר שפקודה 0 תחשב את הערך שלה - היא מחשבת אותו במחזור 11 ולכן המקור הראשון יהיה מוכן במחזור 11 (נשים לב שיש כאן forwarding, פקודה 0 לא עשתה commit לפני מחזור 11).
- המקור השני מוכן עם הקצאת הפקודה - כלומר במחזור 1.
- הערך המידי שמופיע בפקודה הוא 20.
- זמן הביצוע: מכיוון שכל store מפוצל לשתי מיקרו פקודות יהיו שני זמני-ביצוע: ל- STA ו-STD. לכן תמיד נציין שני מס' בתא T exe של פקודת store. ה-STA צריך את הערך של R1 ולכן יכול להיכנס לביצוע כבר במחזור 2. ה-STA צריך את הערך שמגיע במחזור 11 - לכן מתחיל את ביצועו במחזור 12 (כאמור יש כאן forwarding כי זהו אותו מחזור בו המקור מוכן).
- האם יש תנאי-חסימה? זהו store ואף פעם אין עבורו תנאי חסימה! לכן לא ממלאים כלום.
- ה-data של הפקודה מוכן במחזור בו מסתיים החישוב, כלומר במחזור 12.
- אפשר לעשות commit לפקודה במחזור מס' 13 - מחזור אחרי סיום החישוב של STA (STD סיים מוקדם יותר).

פקודה 2: load R3=m[R1+100] (שקף 1/2)

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13
2	load R3=m[R1+100]			110	110	110	1	R1		100	1		2	1	21	22

- מכיוון שמקצים ארבע פקודות בכל מחזור גם כאן T alloc הוא 1.
- לפקודה יש רק מקור אחד - רגיסטר R1 שנדרש בשביל חישוב הכתובת.
- המקור מוכן כבר מהמחזור הראשון ולכן זמן תחילת הביצוע הוא 2.
- הפעם יש תנאי חסימה על ה-load: נסתכל על כל ה-store שלפנינו - במחזור מס' 1 יש store שלא ידעים את הכתובת שלו - לכן זהו תנאי חסימה מסוג' 1 (תקועים על הכתובת).

פקודה 2: load R3=m[R1+100] (שקף 2/2)

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13
2	load R3=m[R1+100]			110	110	110	1	R1		100	1		2	1	21	22

- מתי ה-data יהיה מוכן? תנאי החסימה הוא עד מחזור מס' 12 - רק אז תסתיים פקודות STA. עד אז בכל מחזור בודקים את תנאי החסימה ורואים שה- loadעדיין חסום - כולל במחזור מס' 12. רק במחזור 13 המעבד בודק שוב ומבין שתנאי החסימה התבטל - ולכן במחזור הבא הוא ניגש ל-L1. האם נקבל Miss או hit ב-L1? נשים לב:
  - גודל הבלוק בקאש הוא 80 (הקסדצימלי).
  - עשינו load לכתובת 40
  - ה-store הוא Write no-allocate ולכן עבורו לא הבאנו כלום למטמון.
  - אנחנו רוצים לקרוא מכתובת 110.
  - לכן נקבל cache Miss (מכיוון ש 110Hexadecimal < C0 = Hexadecimal(40+80).

  אז במחזור 14 ניגשים ב-L1 ומקבלים Miss והחל ממחזור זה סופרים שבעה מחזורים ומקבלים את המידע בסוף מחזור 21.
- אפשר לסיים את הפקודה במחזור העוקב - כלומר במחזור 22.

פקודה 3: store m[R1+40]=R3

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13
2	load R3=m[R1+100]			110	110	110	1	R1		100	1		2	1	21	22
3	store m[R1+40]=R3				50	110	1	R1	P2	40	1	21	Std:22 Sta: 2		22	23

- זמן האלוקציה הוא כרגיל 1 (מקצים ארבע פקודות בכל מחזור).
- המקור הראשון של הפקודה הוא R1 (לא עשינו אליו store לכן לא מחכים לאף רגיסטר פיזי).
- המקור השני הוא ערכו של R3 כפי שנקבע ע"י ה-store בשורה מס' 2 - לכן המקור הוא הרגיסטר הפיזיP2 .
- המקור הראשון, R1, מוכן החל מתחילת הריצה והמקור השני מרגע קבלת המידע בפקודה 2 - 21. נשים לב שמכיוון שעשינו forwarding לא מחכים עד ל-commit ואפשר להעביר את הערך כבר בסוף מחזור 21.
- זמן הביצוע של הפקודה: כאמור זה store ולכן מצפים לראות כאן שני מספרים: - אחד עבור תחילת ה-STD ואחד עבור תחילת ה-STA, כל אחד במחזור העוקב למחזור בו המידע שהפקודה מחכה לו מוכן. לכן STA יכנס לחישוב כבר במחזור 2 ו-STD, שחיכה ל-P2 שהתקבל במחזור 21, נכנס לביצוע במחזור 22.
- ה-data ready קורה במחזור 22.
- זמן ה-commit הוא במחזור אחרי זמן קבלת ה-data כלומר מחזור מס' 23.

פקודה 4: add R1=R1+10

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13
2	load R3=m[R1+100]			110	110	110	1	R1		100	1		2	1	21	22
3	store m[R1+40]=R3				50	110	1	R1	P2	40	1	21	Std:22 Sta: 2		22	23
4	add R1=R1+10	20					2	R1		10	2		3			23

בפקודה זו מתחילים את הסבב הבא של האלוקציות.

- פקודה 4 היא החמישית לעבור הקצאה ולכן מוקצית במחזור מס' 2.
- המקור היחיד שלה הוא R1.
- כמו כן לפקודה יש ערך מידי של 10.
- המקור, R1, מוכן במחזור מס' 2 כלומר מיד כשנכנסה.
- במחזור אחרי קבלת המקור היא נכנסת לביצוע, כלומר כבר במחזור 3. נשים לב שהיא עוקפת את שני פקודות ה-store ולמעשה מתבצעת Out Of Order.
- מכיוון ש-commit נעשה in-order הפקודה עושה commit רק במחזור 23 - בתרגיל זה אפשר לעשות commit לכמה פקודות שנרצה במקביל בכל מחזור.

פקודה 5: branch if(R1 < 100) (שקף 1/2)

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13
2	load R3=m[R1+100]			110	110	110	1	R1		100	1		2	1	21	22
3	store m[R1+40]=R3				50	110	1	R1	P2	40	1	21	Std:22 Sta: 2		22	23
4	add R1=R1+10	20					2	R1		10	2		3			23
5	blt if (R1<100)	20	40	110			2	P4		100	3		4			23

- הקפיצה עושה הקצאה במחזור 2 והמקור שלה הוא הערך שיהיה ב- R1לאחר שיחושב ע"י פקודה מס' 4, כלומר P4.
- המקור P4 מוכן כבר במחזור 3 (מחושב במחזור 3 בשורה מס' 4).
- מכיוון שהמקור מוכן כבר במחזור 3 ביצוע הפקודה יכול להתחיל כבר במחזור 4.

פקודה 5: branch if(R1 < 100) (שקף 2/2)

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13
2	load R3=m[R1+100]			110	110	110	1	R1		100	1		2	1	21	22
3	store m[R1+40]=R3				50	110	1	R1	P2	40	1	21	Std:22 Sta: 2		22	23
4	add R1=R1+10	20					2	R1		10	2		3			23
5	blt if (R1<100)	20	40	110			2	P4		100	3		4			23

הערה: נניח ויש בטבלה פקודת קפיצה עבורה מתקבל חיזוי-שגוי (אם כי זה לא המצב בתרגיל) - אם נתון שלוקח חמישה מחזורים עד שמגלים את השגיאה בחיזוי אז אחרי הקפיצה נכנסות פקודות לא נכונות במשך חמישה מחזורים ואנחנו צריכים למלא את הטבלה גם עבורן, אבל הן לא עושות commit - אף פעם לא עושים commit לפקודה לא נכונה!

הנחיה אפשרית בתרגיל:

- ביצוע פקודת branch אורך מחזור אחד.
  - אם החיזוי מתגלה כשגוי, במחזור הבא מבוצע flush (בזמן t+1).
  - הפקודות מהמסלול הנכון מבצעות אלוקציה 5 מחזורים לאחר flush (בזמן t+6).

פקודה 6: load R2=m[R1+30] (שקף 1/3)

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13
2	load R3=m[R1+100]			110	110	110	1	R1		100	1		2	1	21	22
3	store m[R1+40]=R3				50	110	1	R1	P2	40	1	21	Std:22 Sta: 2		22	23
4	add R1=R1+10	20					2	R1		10	2		3			23
5	blt if (R1<100)	20	40	110			2	P4		100	3		4			23
6	load R2=m[R1+30]		110		50	110	2	P4		30	3		4	1, 2	24	25

- הפקודה עושה הקצאה במחזור 2.
- המקור שלה, P4, מוכן במחזור 3 - אותו דבר כמו עבור פקודה מס' 5.
- נשלח את הפקודה לביצוע, כלומר לחישוב כתובת הטעינה, במחזור 4 - המחזור אחרי שהמקור מוכן.

פקודה 6: load R2=m[R1+30] (שקף 2/3)

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:		12	13
2	load R3=m[R1+100]			110	110	110	1	R1		100	1		2	1	21	22
3	store m[R1+40]=R3				50	110	1	R1	P2	40	1	21	Std:22 Sta: 2		22	23
4	add R1=R1+10	20					2	R1		10	2		3			23
5	blt if (R1<100)	20	40	110			2	P4		100	3		4			23
6	load R2=m[R1+30]		110		50	110	2	P4		30	3		4	1, 2	24	25

- האם יש ל-load זה תנאי חסימה? צריך להסתכל על כל ה-stores שלפניו - יש שניים (פקודות 3 ו-1):
- נשים לב שה-load קורא מכתובת 50 ופקודה מס' 3 היא store שכותב לשם - לכן ה-load חסום על תנאי מס' 2 עבורה (תנאי חסימה על ה-data).
- כמו כן יש תנאי חסימה מה-store בשורה מס' 1 בגלל שבמחזור 4 עוד לא סיימנו לחשב את הכתובת של פקודה מס' 1 (פקודת store) ולכן עד מחזור 12 לא ידוע האם גם היא תכתוב לכתובת 50 או לא. זוהי חסימה על הכתובת - תנאי-חסימה מס' 1. בפועל במקרה זה תנאי מס' 1 הוא חלש יותר כי הוא ישתחרר מוקדם יותר אבל זוהי אכן חסימה.

פקודה 6: load R2=m[R1+30] (שקף 3/3)

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13
2	load R3=m[R1+100]			110	110	110	1	R1		100	1		2	1	21	22
3	store m[R1+40]=R3				50	110	1	R1	P2	40	1	21	Std:22 Sta: 2		22	23
4	add R1=R1+10	20					2	R1		10	2		3			23
5	blt if (R1<100)	20	40	110			2	P4		100	3		4			23
6	load R2=m[R1+30]		110		50	110	2	P4		30	3		4	1, 2	24	25

- במחזור 22 מתבטל תנאי 1, ב-23 ה-load נבדק שוב, ובמחזור 24 מתבצע forwarding אליו - המידע מוכן בסוף המחזור בו נעשה forwarding - לכן במחזור 24.
- ניתן יהיה להתחייב (commit) על הפקודה במחזור 25.

תשובה סופית

Src reg:
Pi / Ri:
Store:
Src1: addr
Src2: data

0: ready
1: addr blocking
2: data not ready

Pdst	instruction	R1	R2	R3	addr	data	T alloc	src1	src2	Imm	T src1 ready	T src2 ready	T exe	Load block code	T data ready	T commit
0	load R2=m[R1+30]	10	40	10	40	40	1	R1		30	1		2	0	11	12
1	store m[R2+20]=R1				60	10	1	P0	R1	20	11	1	Std: 2 Sta:12		12	13
2	load R3=m[R1+100]			110	110	110	1	R1		100	1		2	1	21	22
3	store m[R1+40]=R3				50	110	1	R1	P2	40	1	21	Std:22 Sta: 2		22	23
4	add R1=R1+10	20					2	R1		10	2		3			23
5	blt if (R1<100)	20	40	110			2	P4		100	3		4			23
6	load R2=m[R1+30]		110		50	110	2	P4		30	3		4	1, 2	24	25
7	store m[R2+20]=R1				130	20	2	P6	P4	20	24	3	Std:4 Sta:25		25	26
8	load R3=m[R1+100]			120	120	120	3	P4		100	3		4	1	27	28
9	store m[R1+40]=R3				60	120	3	P8	P4	40	27	3	Sta:4 Std:28		28	29
10	add R1=R1+10	30					3	P4		10	3		4			29
11	blt if (R1<100)	30	110	120			3	P10		100	4		5			29