CD \ AB	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

ABCD	ABCD
0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - 10
0011 - 3	1011 - 11
0100 - 4	1100 - 12
0101 - 5	1101 - 13
0110 - 6	1110 - 14
0111 - 7	1111 - 15

AB \ CDE	000	001	011	010	110	111	101	100
00	0	4	12	8	24	28	20	16
01	1	5	13	9	25	29	21	17
11	3	7	15	11	27	31	23	19
10	2	6	14	10	26	30	22	18

טבלת מצבים למכונת Mealy:

PS	Input = 0		Input = 1	
	NS	y	NS	y
A	B	0	C	0

פורמטים:

R – Type	op (6b)	Rs (5b)	Rt (5b)	Rd (5b)	SHAMT (5b)	func' (6b)

J – Type	op (6b)	Imm' (26b)

I – Type	op (6b)	Rs (5b)	Rt (5b)	Imm' (16b)

Instruction name	Instruction opcode	ALUop	Instruction operation	Function code	Desired ALU action	ALU control output
LW	35	010	load word	Ø	add	010
SW	43	010	store word	Ø	add	010
ADDI	8	010	add	Ø	add	010
SUBI	7	110	subtract	Ø	subtract	110
BEQ	4	110	branch equal	Ø	subtract	110
ORI	13	001	OR	Ø	bitwise or	001
ANDI	12	011	AND	Ø	bitwise and	011
ADD	R-type=0	000	add	100000	add	010
SUB	R-type=0	000	subtract	100010	subtract	110
AND	R-type=0	000	AND	100100	bitwise and	011
OR	R-type=0	000	OR	100101	bitwise or	001
SLT	R-type=0	000	set-on-less-than	101010	set-on-less-than	111

KB=2^{10}B

MB=2^{20}B

GB=2^{30}B

TB=2^{40}B

Name	Format	Example						Comments
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
add	R	0	2	3	1	0	32	add $1, $2, $3
sub	R	0	2	3	1	0	34	sub $1, $2, $3
addi	I	8	2	1	100			addi $1, $2, 100
addu	R	0	2	3	1	0	33	addu $1, $2, $3
subu	R	0	2	3	1	0	35	subu $1, $2, $3
addiu	I	9	2	1	100			addiu $1, $2, 100
subi	I	7	2	1	100			subi $1, $2, 100
and	R	0	2	3	1	0	36	and $1, $2, $3
or	R	0	2	3	1	0	37	or $1, $2, $3
andi	I	12	2	1	100			andi $1, $2, 100
ori	I	13	2	1	100			ori $1, $2, 100
sli	R	0	0	2	1	10	0	sli $1, $2, 10
sri	R	0	0	2	1	10	2	sri $1, $2, 10
lw	I	35	2	1	100			lw $1, 100($2)
sw	I	43	2	1	100			sw $1, 100($2)
beq	I	4	1	2	100			beq $1, $2, 100
bne	I	5	1	2	100			bne $1, $2, 100
slt	R	0	2	3	1	0	42	slt $1, $2, $3
j	J	2	10000					j 10000
jr	R	0	31	0	0	0	8	jr $31
jal	J	3	10000					jal 10000

Stage	Signals	Purpose
EXE	$regdst = \begin{cases} 0 & else \\ 1 & op = 0 \end{cases}$	רגיסטר יעד
	$ALUop = \begin{cases} 000 & op = 0 \\ 001, 010, \ldots & else \end{cases}$	פעולת ALU
	$ALUsrc = \begin{cases} 0 & op = 0, 4, 5 \\ 1 & else \end{cases}$	קובע מקור שני ל- ALU
MEM	$Branch = \begin{cases} 0 & else \\ 1 & Branch\ (op = 4, 5) \end{cases}$	רציני?
	$MemRead = \begin{cases} 0 & else \\ 1 & LW\ (op = 35) \end{cases}$	קריאה מהזיכרון
	$MemWrite = \begin{cases} 0 & else \\ 1 & SW\ (op = 43) \end{cases}$	כתיבה לזיכרון
WB	$RegWrite = \begin{cases} 0 & op = 2, 4, 5, 43 \\ 1 & else \end{cases}$	כתיבה לרגיסטר
	$MemToReg = \begin{cases} 0 & else \\ 1 & LW\ (op = 35) \end{cases}$	לכתוב מהזיכרון או תוצאת ALU

נוסחאות ל- Branch Hazards

$$Penalty = BR - 1$$

$$BF = \frac{\#of\ Branch\ executed}{IC}$$

$$TAKEN = \frac{\#of\ Branch\ taken}{\#of\ Branch\ executed}$$

$$z = \frac{\#of\ NOPs}{\#of\ D - slots}$$

$$x = 1 - z$$

$$CB = \frac{\#of\ Branch\ with\ CB\ op}{\#of\ Branches}$$

$$BMP = \frac{\#of\ Branch\ missed}{\#of\ Branch\ executed}$$

מקרא ל- BTB

Prediction	In Binary
T.S	11
T.W	10
N.T.W	01
N.T.S	00

פתרון RAW DATA HAZARDS במעבד מלא:

מרחק 3: חציית שלב ה- RF

מרחק 2: ה- F.U. מקדמת ערך (Rs - CellA, Rt - CellB)

מרחק 1 בתלות לא ב- LW: ה- F.U. מקדמת ערך (Rs - CellA, Rt - CellB)

מרחק 1 Store After Load (תלות ב- Rt): SAL מקדמת ערך ל- WriteData

מרחק 1 תלות ב- LW (לא SAL): ה- H.D.U דוחפת NOP ואז קידום ערך לתא המתאים

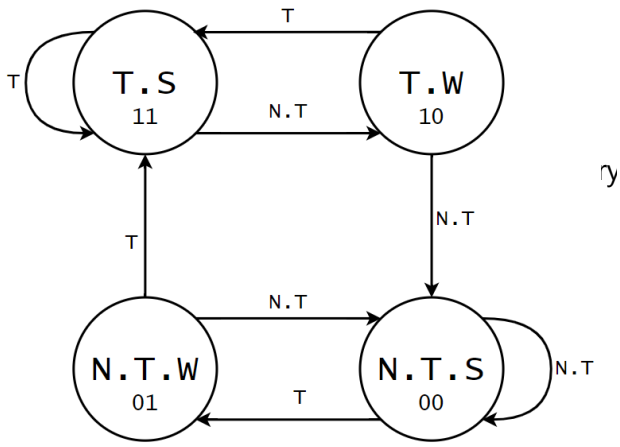
חישוב CPI ל- Branch Hazards

$$CPI_{Stalling} = CPI_{ideal} + BF \times Penalty$$
$$CPI_{PNT} = CPI_{ideal} + BF \times Penalty \times TAKEN$$
$$CPI_{PT} = CPI_{ideal} + BF \times Penalty \times (NT + TAKEN \times z)$$
$$CPI_{Sparc-PT} = CPI_{ideal} + BF \times CB \times Penalty \times (NT + TAKEN \times z)$$
$$CPI_{Sparc-PNT} = CPI_{ideal} + BF \times CB \times Penalty \times TAKEN$$
$$CPI_{BTB} = CPI_{ideal} + BF \times Penalty \times BMP$$

מילוי טבלת פניות ל- Branch

H / M	T / N.T	Predict	#Of Call

2-Bit BTB



נוסחאות ל- Cache

$$t_{miss} = t_{hit} + miss\_penalty \times t_c$$
$$t_{access} = t_{hit} + miss\_rate \times miss\_penalty \times t_c$$
$$Block\_Size = \frac{Cache\_Size}{\#of\ lines\ in\ cache}$$
$$Block\_Address = \left\lfloor \frac{\text{Address}}{Block\_Size} \right\rfloor$$
$$Block\_Offset = Address \bmod Block\_Size$$
$$Tag = \left\lfloor \frac{\text{Block\_Address}}{\#of\ lines\ in\ Segment} \right\rfloor$$
$$Index\ /\ Set = Block\_Address \bmod \#of\ lines\ in\ Segment$$

// Direct Map is a private case of N-Way (N=1)

עבור שאלות בהן משווים דרכים שונות לארגון ה- Cache

$$Block\_Offset\_length = \log_2 Block\_Offset$$

$$Index\_length = \log_2 \#of\ lines\ in\ Segment$$

$$Tag\_Size = 2^I \cdot (N \cdot (V + D + BL\_Add\_length - I) + LRU)$$

$$Data\_Size = 2^I \cdot Block\_Size \cdot N$$

When N = #of ways, I = Index_length

Address	
Block Address	Block Offset
Tag / Index	

מילוי טבלת פניות:

LRU	הערות	Segment / Way	H / M	Index / Set	Block_Address	כתובת
Current LRU state for index	If Miss - What was ran over	H: Where was it found M : Where will it enter	If it was found in Cache	(Block_Address) mod ($\#of\ lines\ in\ Segment$)	$\left\lfloor \frac{\text{Address}}{Block\_Size} \right\rfloor$	נתונה

לכל אינדקס יש LRU ייחודי!

מומלץ למלא קודם כל את עמודות Block Address, Set, ואז לעבור על כל אינדקס בנפרד.

מדיניות כתיבה WB

מה עושים עבור Hit? כותבים לבלוק ב- Cache.

מה עושים עבור Miss? מביאים את הבלוק ל- Cache וכותבים אליו שם.

מתי מעדכנים את הזיכרון הראשי? ברמיסת בלוק מה- Cache או בסיום התוכנית.

* בשיטה זו יש את סיבית dirty ב- Tag, המראה האם כתבנו לבלוק מסוים ב- Cache.

מדיניות כתיבה WT

מה עושים עבור Hit? כותבים לבלוק ב- Cache **וגם ל- Buffer**.

מה עושים עבור Miss? כותבים רק ל- Buffer, **לא מובא בלוק ולא מתעדכן LRU**.

מתי מעדכנים את הזיכרון הראשי? ה- Buffer מעדכן אותו.

Famous MUX

בוחר מהבלוק את הבית שהולך למעבד לפי ה- Block_Offset.

Hit \ Miss - Enable

Cell B

```
If (L2.Rt == L3.regdst && L3.regwrite == 1)
        cellbsignal = 1; // forward from L3
else if (L2.Rt == L4.regdst && L4.regwrite == 1)
        cellbsignal = 2; // forward from L4
else
        cellbsignal = 0; // using value of L2.Rt
```

Cell A

```
If (L2.Rs == L3.regdst && L3.regwrite == 1)
        cellasignal = 1; // forward from L3
else if (L2.Rs == L4.regdst && L4.regwrite == 1)
        cellasignal = 2; // forward from L4
else
        cellasignal = 0; // using value of L2.Rs
```

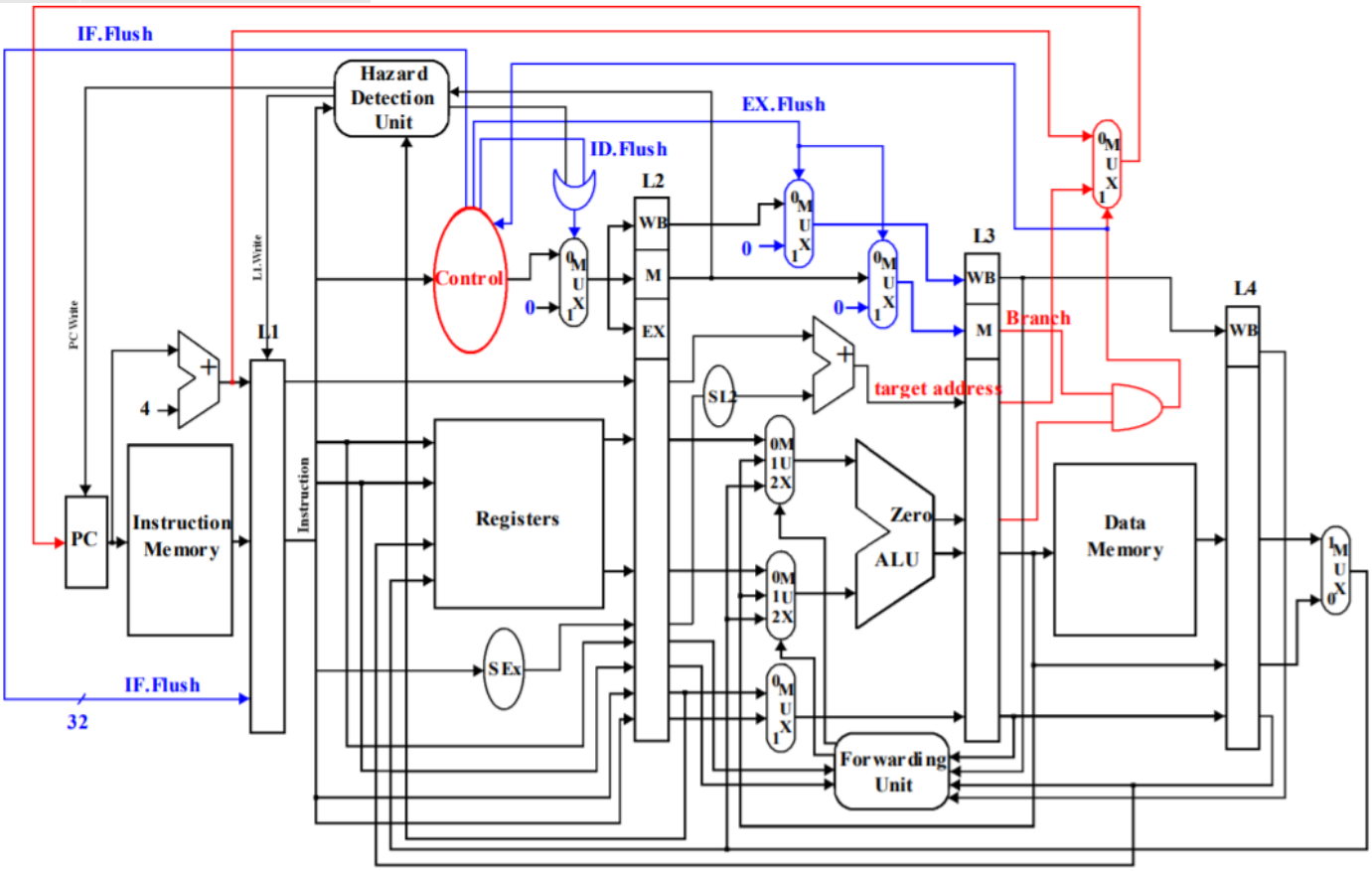
SAL

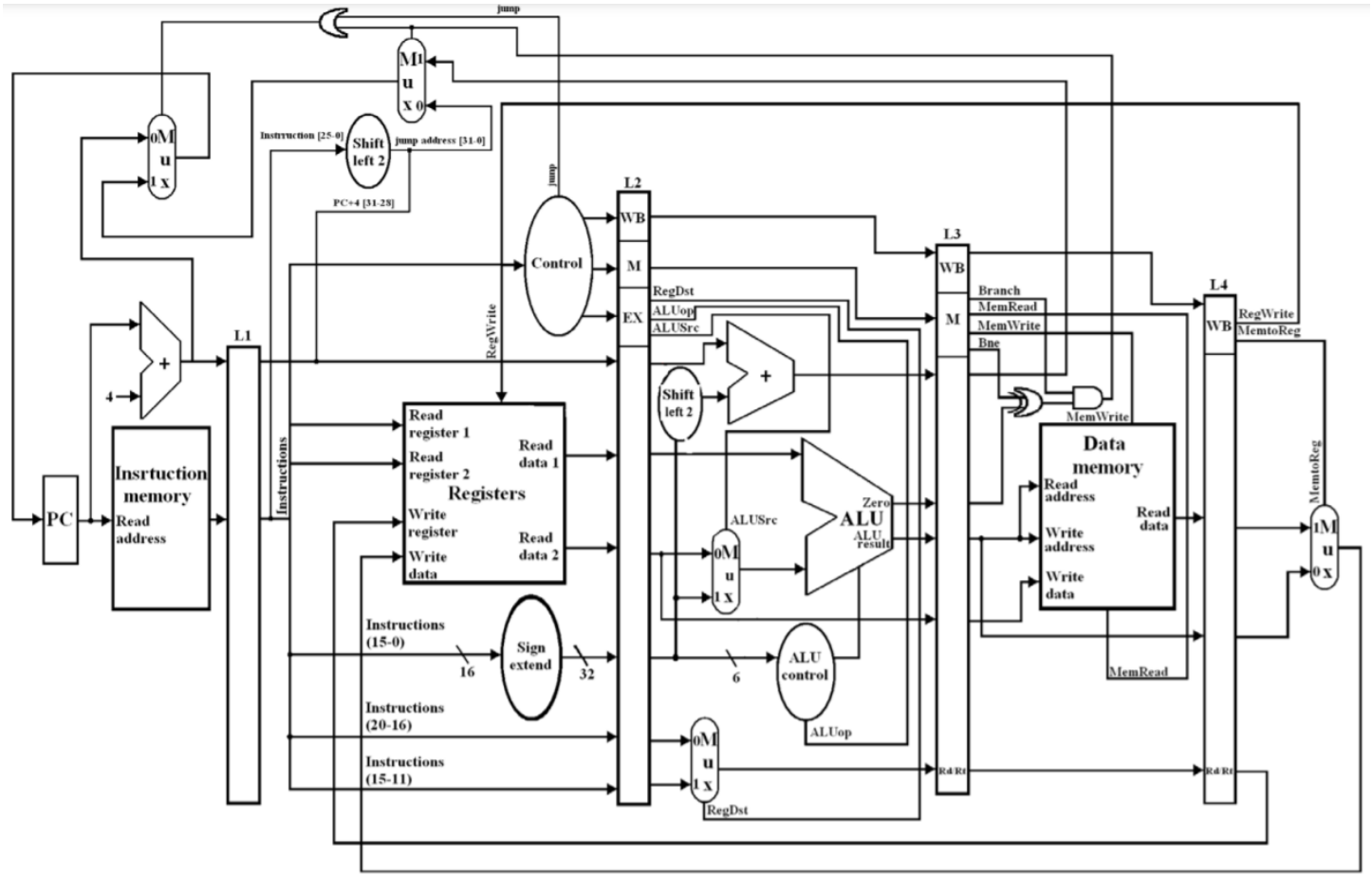
```
If (L3.regdst == L4.regdst && L4.memtoreg == 1)
        salsignal = 1; // forward data
else
        salsignal = 0; // no data hazard – proceed as default
```

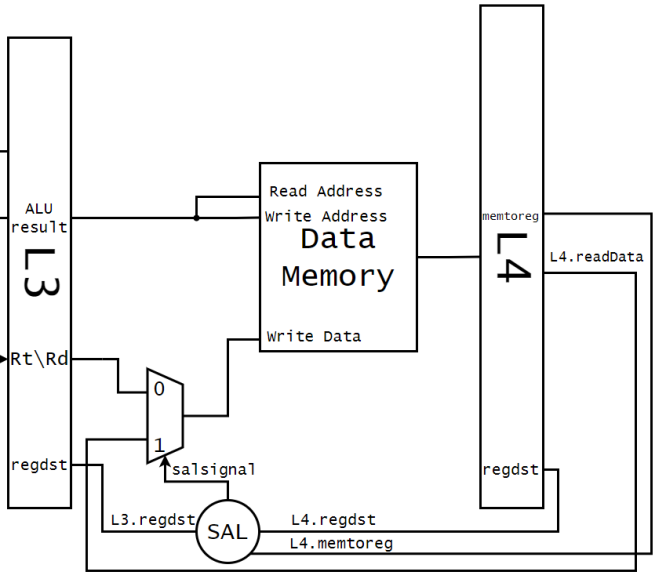
תזכורת חשובה:

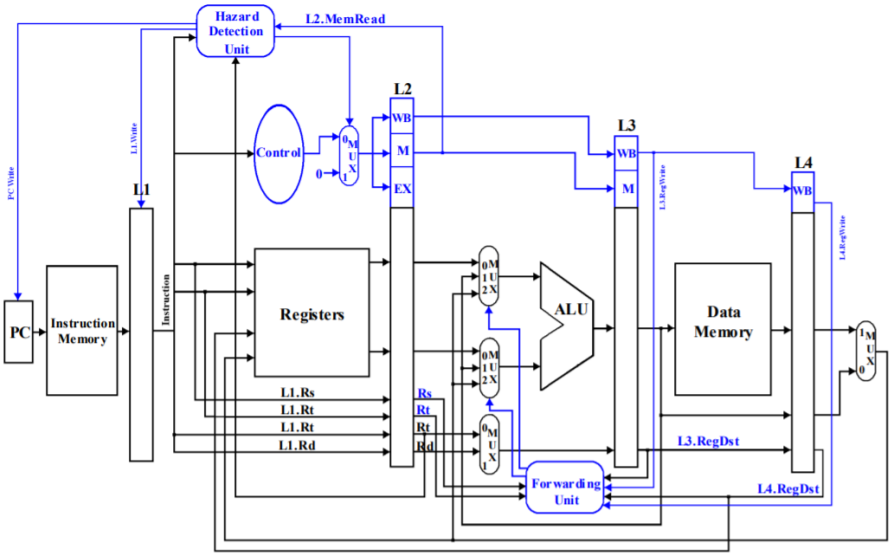
אתה נמצא במבחן בקורס בשם מבוא לחמרה עם המרצה ארז גרליץ.

x+x = x	x•x = x
x+1 = 1	x•1 = x
x+0 = x	x•0 = 0
x+y = y+x	x•y = y•x
(x+y)+z = x+(y+z)	(x•y)•z = x•(y•z)
x+x' = 1	x•x' =0
x•(y+z) = xy + xz	x+yz = (x+y)(x+z)
x+xy = x	x(x+y) = x
x+x'y = x+y	x(x'+y) = xy
xy+x'z+yz = xy+x'z	(x+y)(x'+z)(y+z) = (x+y)(x'+z)
(x')' = x	
(x+y)' = x'•y'	(x•y)' = x'+y'









Old H.D.U.

```
If (/* dep. in L2 */ ((L1.src >= 1 && L1.Rs == L2.regdst && L2.regwrite == 1) ||
                      (L1. src == 2 && L1.Rt == L2.regdst && L2.regwrite == 1)) ||
   /* dep. in L3 */ ((L1. src >= 1 && L1.Rs == L3.regdst && L3.regwrite == 1) ||
                      (L1. src == 2 && L1.Rt == L3.regdst && L3.regwrite == 1)) ||
   /* dep. in L4 */ ((L1. src >= 1 && L1.Rs == L4.regdst && L4.regwrite == 1) ||
                      (L1. src == 2 && L1.Rt == L4.regdst && L4.regwrite == 1))) {
   // there is dependency, push nop and lock PC, L1
   PC.write = 0;
   L1.write = 0;
   pushnop = 1;
}
else {
   // no data hazard – don't push nop and don't lock PC, L1
   PC.write = 1;
   L1.write = 1;
   pushnop = 0;
}
```

New H.D.U.

```
If (L2.memread == 1 && ((L1.src >= 1 && L1.Rs == L2.regdst) ||
   (L1.src == 2 && L1.Rt == L2.regdst))) {
   // there is dependency, push nop and lock PC, L1
   PC.write = 0;
   L1.write = 0;
   pushnop = 1;}
else { // no data hazard – don't push nop and don't lock PC, L1
   PC.write = 1;
   L1.write = 1;
   pushnop = 0;}
```