נספח
תקציר נתונים – ארגון המחשב ושפת סף
ביצועי המעבד

סימון	פירוש	יחידות	מונח
CCT	Clock cycle time	[sec/cycle]	זמן מחזור שעון
CR	Clock rate	[cycle/sec]	תדר השעון Hz=[1/sec]
CC	Clock cycle (per program)	[cycle/program]	מספר מחזורי שעון בתוכנית
CPI	Clock per instruction	[cycle/ins]	מספר מחזורי שעון לפקודת מכונה
IC	Instruction count	[ins/program]	מספר פקודות מכונה בתוכנית (בריצה)
CPU_T	CPU time (Rum time)	[sec/program]	זמן ריצה של תוכנית
Speedup	בשביל לקבל האצה צריך להיות גדול מ־1	יחס, אין יחידות	מדד ההאצה
MIPS	Million Instructions Per Second	[MIPS]	מיליון פקודות בשנייה

נוסחאות	יחידות	מונח
$CPI = \frac{CC(Clock\ Cycle)}{IC(Instruction\ count)} = \sum_{i=1}^{n} CPI_i \times w_{i=}$	[cycle/ins]	מספר ממוצע של מחזורי שעון לפקודת מכונה
$\boldsymbol{CC\left[\frac{cycle}{program}\right] = IC\left[\frac{ins}{program}\right] \times CPI\left[\frac{cycle}{ins}\right]}$	[cycle/program]	מספר מחזורי שעון בתוכנית
$CPUtime\ \left[\frac{sec}{prog}\right] = IC\ \left[\frac{ins}{program}\right] \times CPI\ \left[\frac{cycle}{ins}\right] \times CCT\ \left[\frac{sec}{cycle}\right]$ $CPUtime\ \left[\frac{sec}{prog}\right] = \frac{IC\ \left[\frac{ins}{program}\right] \times CPI\ \left[\frac{cycle}{ins}\right]}{CR\ \left[\frac{cycle}{sec}\right]}$	$\left[\frac{sec}{prog}\right]$	זמן ריצה של תוכנית
$Speedup = \frac{CPUTime_{slow}}{CPUTime_{fast}}$	יחס חסר יחידות	מדד ההאצה
$CPUTime_{Fast} = CPUTime_{slow}\ x\ [(1 - Fraction + \frac{Fraction}{Speedup}]$ $Speedup_{Total} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$	יחס חסר יחידות	כלל אמדל השפעה של שיפור חלק מהמערכת על כלל המערכת
$MIPS = \frac{IC}{CPUTime\ x\ 10^6} = \frac{CR}{CPI\ x\ 10^6}$	[MIPS]	מדד MIPS

הערה: בחלק מהתרגילים מסומנת יחידת מחזור השעון ב־cycle ובחלק ב־Cc (clock cycle). שימו לב לא להתבלבל עם המונח CC שמשמעו מספר מחזורי שעון **בתוכנית**.

תזכורת גדלים

חזקת 10	סימון	שם	חזקת 10	סימון	שם
10^3	K	kilo	10^{-3}	m	milli
10^6	M	mega	10^{-6}	µ	micro
10^9	G	giga	10^{-9}	n	nano
10^{12}	T	terra	10^{-12}	p	pico

ייצוג מידע במחשב

מעברי בסיס שימושיים (ברירת המחדל) מבסיס 10 לבסיסים 2, 4, 8, 16

10	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000
4	0	1	2	3	10	11	12	13	20	21	22	23	30	31	32	33	40
8	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20
16	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10

n=4	ללא סימן	ערך מוחלט סימן	משלים ל־2	n=4	ללא סימן	ערך מוחלט סימן	משלים ל־2
0000	0	0	0	1000	8	-0	-8
0001	1	1	1	1001	9	-1	-7
0010	2	2	2	1010	10	-2	-6
0011	3	3	3	1011	11	-3	-5
0100	4	4	4	1100	12	-4	-4
0101	5	5	5	1101	13	-5	-3
0110	6	6	6	1110	14	-6	-2
0111	7	7	7	1111	15	-7	-1

נקודות חשובות בשיטת ייצוג מספרים משלים ל־2

- שיטה זו היא השימושית ביותר בעולם המחשבים לייצוג מספרים עם סימן. היתרון המרכזי של שיטת משלים ל־2 בייצוג מספרים שלמים עם סימן הוא בכך שתמיכת החומרה פשוטה, ולכן מהירה יותר. (אלגוריתם החיבור במשלים ל־2 וללא סימן זהה.)

- הקריטריון לגלישה אריתמטית מתחום הייצוג ב־n סיביות הוא C_n xor C_{n-1}=overflow.

- תהליך הרחבת סימן משלים ל־2 נקרא signed extend, או בקיצור .S.E. יתבצע ריפוד סיבית הסימן לחלק המורחב. (שכפול אפסים או אחדים בהתאם לערכו של ה־MSB לפני ההרחבה לחלק המורחב.)

- אם נרצה לבצע הרחבת סימן בייצוג בשיטת המספרים ללא סימן, הפעולה תהיה תמיד הוספת אפסים לחלק המורחב. פעולה זו מכונה zero extend, או בקיצור .Z.E.

שיטת הנקודה הצפה

בשנת 1985 פורסם תקן 754 שקבע איגוד המהנדסים הבין-לאומי IEEE (Institute of Electrical and Electronics Engineers). התקן נקרא **IEEE 754**, והוא **התקן המקובל בעולם המחשבים החל משנת 1997.**

תקן זה כולל שתי צורות ייצוג: **דיוק יחיד** (single precision) **ודיוק כפול** (double precision). עבור דיוק יחיד משתמשים ב- 32 סיביות, ועבור דיוק כפול משתמשים ב- 64 סיביות. אנו נתמקד בשיטת הדיוק היחיד, שבה מחולקות הסיביות באופן הזה:

- סיביות 0-22 (23 סיביות) משמשות כשדה המנטיסה; **mantissa** (בספר הקורס שדה זה מכונה בשם **fraction** יש ספרים המכנים שדה זה בשם significant)
- סיביות 23-30 (8 סיביות) משמשות כשדה החזקה; **exponent**
- סיבית 31 משמשת כשדה הסימן; **sign**

בשדה המנטיסה מאוחסנת המנטיסה של הייצוג המנורמל. 23 הסיביות המופיעות לאחר הנקודה נרשמות בשדה זה, משמאל לימין. לא לשכוח שלמעשה יש **24** סיביות משום שה – Hidden Bit (הסיבית הנחבאת) שערכה תמיד 1 בהצגה מנורמלת אינה מוצגת בתקן IEE754.

בשדה החזקה מאוחסן ערך החזקה. הערכים 0 ו-255 בשדה החזקה שמורים לייצוגים מיוחדים. המספר 0 מיוצג על-ידי 32 סיביות שערכן 0. ייצוגים מיוחדים נוספים שמורים, למשל, עבור $+\infty$ (כלומר, מספרים חיוביים שערכם גבוה מן הערך הגבוה ביותר הניתן לייצוג) ו- $-\infty$ (כלומר, מספרים שליליים שערכם המוחלט גבוה מן הערך המוחלט הגבוה ביותר הניתן לייצוג).

דוגמה:

רשמו את הייצוג של המספר –1.5 בתקן IEEE 754, בייצוג של דיוק יחיד.

פתרון

1. נרשום את –1.5 בצורה מנורמלת:

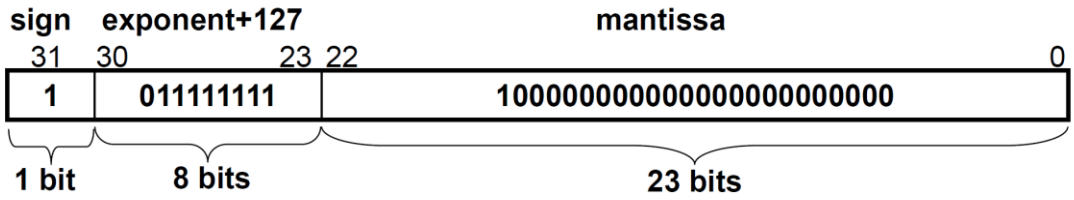
$$-1.5 = -2^0 \cdot (1.1)_2$$

נזכור שאת הסיבית 1 שמשמאל לנקודה אנחנו לא מציגים (הסיבית הנחבאת – Hidden Bit) אלא רק את 23 הסיביות של המנטיסה שמימין לנקודה.

2. נרשום את ערכי השדות:

mantissa =(1.) 10000000000000000000000
exponent = 0 + bias = 0 +127 = 01111111
sign = 1

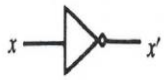
והאוגר נראה כך:



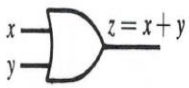
אלגברה בוליאנית

	חוק/משפט	עבור OR (logical sum)	עבור AND (logical product)
א	Identity Law(איבר היחידה)	A + 0 = A	A · 1 = A
ב	Idempotence (אידמפוטנט)	A + A= A	A · A = A
ג	Annihilation (איון)	A + 1= 1	A · 0 = 0
ד	Inverse Law (הופכי)	A + A' = 1	A · A' = 0
ה	Commutative Law (חילוף)	A + B = B + A	A· B = B · A
ו	Associative Law (קיבוץ)	A + (B+C) = (A+B) + C	A· (B · C) = (A· B) · C
ז	Distributive Law (פילוג)	A· (B+C) = (A· B) + (A· C)	A + (B · C) = (A+B) · (A+C)
ח	Involution Law (הופכי כפול)	(A')'=A	
ט	DeMorgan's Theorem משפט דה־מורגן	(A + B)' = A' · B'	(A · B)' = A' + B'
י	Absorption Law (צמצום)	A + (A · B) = A	A · (A+B) = A
יא	Disappearing Opposite (ההופכי הנעלם)	A + (A'·B) = A+B	A· (A'+B) = A·B

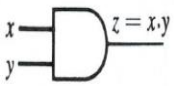
שערים לוגיים



3. שער NOT (או מהפך)

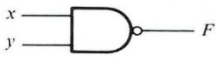


2. שער OR בעל שתי כניסות



1. שער AND בעל שתי כניסות

NAND



$F = (xy)'$

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

NOR

$F = (x + y)'$

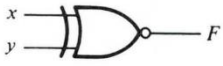
x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

Exclusive-OR (XOR)

$F = xy' + x'y$
$= x \oplus y$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR or equivalence



$F = xy + x'y'$
$= x \odot y$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

פקודות של ה־MIPS מפורמט R (opcode=0 000000b)

	מבצעת	שם	פקודת אסמבלי	הפעולה	קידוד פקודת מכונה	FUNC
אריתמטי	חיבור בין אוגרים	add	add $rd,$rs,$rt	$rd=$rs+$rt	000000,sssss,ttttt,ddddd,00000,100000	32=0x20
	חיבור. מתעלמת מגלישה אריתמטית	addu	addu $rd,$rs,$rt	$rd=$rs+$rt	000000,sssss,ttttt,ddddd,00000,100001	33=0x21
	חיסור בין אוגרים	sub	sub $rd,$rs,$rt	$rd=$rs-$rt	000000,sssss,ttttt,ddddd,00000,100010	34=0x22
	חיסור. מתעלמת מגלישה אריתמטית	subu	subu $rd,$rs,$rt	$rd=$rs-$rt	000000,sssss,ttttt,ddddd,00000,100011	35=0x23
	כפל במשלים לשתיים	multiply	mult $rs,$rt	{hi,lo}=$rs*$rt	000000,sssss,ttttt,00000,00000,011000	24=0x18
	כפל ללא סימן	multiply unsign	multu $rs,$rt	{hi,lo}=$rs*$rt	000000,sssss,ttttt,00000,00000,011001	25=0x19
	חילוק במשלים לשתיים	divide	div $rs,$rt	lo= $rs/$rt hi=$rs%$rt	000000,sssss,ttttt,00000,00000,011010	26=0x1A
	חילוק ללא סימן	divide unsign	divu $rs,$rt	lo= $rs/$rt hi=$rs%$rt	000000,sssss,ttttt,00000,00000,011011	27=0x1B
	העתקת הערך של lo לאוגר $rd	move from lo	mflo $rd	$rd=lo	000000,00000,00000,ddddd,00000,010010	18=0x12
	העתקת הערך של hi לאוגר $rd	move from hi	mfhi $rd	$rd=hi	000000,00000,00000,ddddd,00000,010000	16=0x10
לוגי	פעולת "וגם" לוגית	and	and $rd,$rs,$rt	$rd=$rs&$rt	000000,sssss,ttttt,ddddd,00000,100100	36=0x24
	פעולת "או" לוגית	or	or $rd,$rs,$rt	$rd=$rs|$rt	000000,sssss,ttttt,ddddd,00000,100101	37=0x25
	פעולת "שוני" לוגית	xor	xor $rd,$rs,$rt	$rd=$rs⊕ $rt	000000,sssss,ttttt,ddddd,00000,100110	38=0x26
	פעולת "לא או" לוגית	nor	nor $rd,$rs,$rt	$rd=$rs ↓ $rt	000000,sssss,ttttt,ddddd,00000,100111	39=0x27
תנאי	אם $rt > $rs לפי משלים לשתיים אז $rd מקבל 1 (אמת), אחרת 0 (שקר)	Set on less than	slt $rd,$rs,$rt	If $rs<$rt than $rd=1 else $rd=0	000000,sssss,ttttt,ddddd,00000,101010	42=0x2A
	אם $rt > $rs לפי ללא סימן אז $rd מקבל 1 (אמת), אחרת 0 (שקר)	Set on less than unsigned	sltu $rd,$rs,$rt	If $rs<$rt than $rd=1 else $rd=0	000000,sssss,ttttt,ddddd,00000,101011	43=0x2B
הזזה	הזזה שמאלה וריפוד באפסים מימין	shift left logical	sll $rd,$rt,shift	$rd=$rt<<(shift)	000000,00000,ttttt,ddddd,shshs,000000	0=0x00
	הזזה ימינה וריפוד באפסים משמאל	shift right logical	srl $rd,$rt,shift	$rd=$rt>>(shift)	000000,00000,ttttt,ddddd,shshs,000010	2=0x02
	הזזה ימינה וריפוד משמאל בסיבית הסימן	shift right arithmetic	sra $rd,$rt,shift	$rd=$rt>>(shift) With sign bit	000000,00000,ttttt,ddddd,shshs,000011	3=0x03
ניתוב בקרה	קפיצה ללא תנאי לכתובת האוגר $rs	jump register	jr $rs	PC=$rs	000000,sssss,00000,00000,00000,001000	8=0x08
	קפיצה ללא תנאי לכתובת האוגר $rs ושמירת כתובת חזרה באוגר $ra	jump and link register	jalr $rs	PC=$rs $ra=PC+4	000000,sssss,00000,11111,00000,001001	9=0x09
	עצירת התוכנית	break	break	Stop program	000000,00000,00000,00000,00000,001101	13=0x0D

פקודות של ה־MIPS מפורמט I (I - 16 imm הסיביות הנמוכות בקידוד הפקודה 0.15)

	מבצעת	שם	פקודת אסמבלי	הפעולה	קידוד פקודת מכונה	opcode	ריפוד
אריתמטי	חיבור עם קבוע	add immediate	addi $rt,$rs,imm	$rt=$rs+imm	001000,sssss,ttttt,iiiiiiiiiiiiiii	8=0x08	SE
	חיבור עם קבוע מתעלמת מגלישה אריתמטית	add immediate unsigned	addiu $rt,$rs,imm	$rt=$rs+imm	001001,sssss,ttttt,iiiiiiiiiiiiiii	9=0x09	SE
לוגי	פעולת "וגם" לוגית	and immediate	andi $rt,$rs,imm	$rt=$rs&imm	001100,sssss,ttttt,iiiiiiiiiiiiiii	12=0x0C	ZE
	פעולת "או" לוגית	or immediate	ori $rt,$rs,imm	$rt=$rs|imm	001101,sssss,ttttt,iiiiiiiiiiiiiii	13=0x0D	ZE
	פעולת "שוני" לוגית	xor immediate	xori $rt,$rs,imm	$rt=$rs⊕imm	001110,sssss,ttttt,iiiiiiiiiiiiiii	14=0x0E	ZE
תנאי	אם imm > $rs לפי משלים לשתיים, אז $rt מקבל 1, אחרת 0	Set on less than immediate	slti $rt,$rs,imm	If $rs<imm than $rt=1 else $rt=0	001010,sssss,ttttt,iiiiiiiiiiiiiii	10=0x0A	SE
	אם imm> $rs לפי ללא סימן אז $rt מקבל 1, אחרת 0	Set on less than immediate unsigned	sltiu $rt,$rs,imm[1]	If $rs<imm than $rt=1 else $rt=0	001011,sssss,ttttt,iiiiiiiiiiiiiii	11=0x0B	SE
ניתוב בקרה על תנאי	קפיצה אם $rs=$rt	branch on equal	beq $rs,$rt,imm[1]	If $rs=$rt than pc=pc+4+imm*4	000100,sssss,ttttt,iiiiiiiiiiiiiii	4=0x04	SE
	קפיצה אם $rs≠$rt	branch on not equal	bne $rs,$rt,imm[1]	If $rs≠$rt than pc=pc+4+imm*4	000101,sssss,ttttt,iiiiiiiiiiiiiii	5=0x05	SE
	קפיצה אם $rs ≤0	branch on less than or equal zero	blez $rs,imm[1]	If $rs ≤0 than pc=pc+4+imm*4	000110,sssss,00000,iiiiiiiiiiiiiii	6=0x06	SE
	קפיצה אם $rs>0	branch on greater than zero	bgtz $rs,imm[1]	If $rs > 0 than pc=pc+4+imm*4	000111,sssss,00000,iiiiiiiiiiiiiii	7=0x07	SE
גישה לזיכרון טעינה	טעינת בית עם סימן מכתובת $rs+imm בזיכרון	load byte	lb $rt,imm($rs)	$rt=mem($rs+imm)[2]	100000,sssss,ttttt,iiiiiiiiiiiiiii	32=0x20	SE
	טעינת בית ללא סימן מכתובת $rs+imm בזיכרון	load byte unsigned	lbu $rt,imm($rs)	$rt=mem($rs+imm)[3]	100100,sssss,ttttt,iiiiiiiiiiiiiii	36=0x24	SE
	טעינת חצי מילה סימן מכתובת $rs+imm בזיכרון	load half word	lh $rt,imm($rs)	$rt=mem($rs+imm)[4]	100001,sssss,ttttt,iiiiiiiiiiiiiii	33=0x21	SE
	טעינת חצי מילה ללא סימן מכתובת $rs+imm בזיכרון	load half word unsigned	lhu $rt,imm($rs)	$rt=mem($rs+imm)[5]	100101,sssss,ttttt,iiiiiiiiiiiiiii	37=0x25	SE
	טעינת מילה מכתובת $rs+imm בזיכרון	load word	lw $rt,imm($rs)	$rt=mem($rs+imm)	100011,sssss,ttttt,iiiiiiiiiiiiiii	35=0x23	SE
גישה לזיכרון שמירה	שמירת הבית הנמוך ב$rt במקום $rs+imm בזיכרון	store byte	sb $rt,imm($rs)	mem($rs+imm)=$rt	101000,sssss,ttttt,iiiiiiiiiiiiiii	40=0x28	SE
	שמירת חצי המילה הנמוכה ב־rt במקום $rs+imm בזיכרון	Store half word	sh $rt,imm($rs)	mem($rs+imm)=$rt	101001,sssss,ttttt,iiiiiiiiiiiiiii	41=0x29	SE
	שמירת $rt במקום $rs+imm בזיכרון	Store word	sw $rt,imm($rs)	mem($rs+imm)=$rt	101011,sssss,ttttt,iiiiiiiiiiiiiii	43=0x2B	SE
טעינת קבוע	הכנסת הערך המיידי ל־16 הסיביות הגבוהות של $rt ואיפוס 16 הסיביות הנמוכות של $rt	Load upper immediate	lui $rt,imm	$rt=imm *$2^{16}$	001111,00000,ttttt,iiiiiiiiiiiiiii	15=0x0F	-

1. בעת כתיבת הפקודה באסמבלי, נרשום תווית label שלשם תתבצע הקפיצה. האסמבלר מחשב את נוסחת ה־branch target ומקודד את שדה ה־imm.
2. הבית מהזיכרון נטען לבית הנמוך (0–7) ב־$rt, והסיביות הגבוהות (8–31) עוברות S.E.
3. הבית מהזיכרון נטען לבית הנמוך (0–7) ב־$rt, והסיביות הגבוהות (8–31) עוברות Z.E.
4. חצי המילה מהזיכרון נטענת לחצי המילה הנמוכה (0–15) ב־$rt, והסיביות הגבוהות (16–31) עוברות S.E.
5. חצי המילה מהזיכרון נטענת לחצי המילה הנמוכה (0–15) ב־$rt, והסיביות הגבוהות (16–31) עוברות Z.E.

פקודות של ה־MIPS מפורמט J

מבצעת	שם	פקודת אסמבלי	הפעולה	קידוד פקודת מכונה	opcode
קפיצה ללא תנאי לכתובת תווית	jump	j imm[1]	pc=pc+4[28-31] +imm(0..26)*4	000010iiiiiiiiiiiiiiiiiiiiiiii	2=0x02
קפיצה לכתובת תווית ושמירת pc+4 באוגר $ra	jump and link	jal imm[1]	pc=pc+4[28-31] +imm(0..26)*4 and $ra=pc+4 ($ra=$31)	000011iiiiiiiiiiiiiiiiiiiiiiii	3=0x03

1. בעת כתיבת הפקודה באסמבלי, נרשום תווית label שלשם תתבצע הקפיצה רק בזמן ריצה, קטע הקוד של מערכת ההפעלה הטוען את התוכנית לזיכרון מחשב את הכתובת של התווית שביחס אליה יש לעדכן את 26 הסיביות של ה־imm בקידוד הפקודה.

הפקודה mul הינה בפורמט מיוחד. הפורמט מזכיר פורמט R (אולם שדה ה opcode שונה מ-0), opcode=0x1c func=0x02. פקודה זו נוספה לגרסאות מתקדמות יותר של המעבד לכפל מקוצר. הפקודה **mul $rd,$rs,$rt** מבצעת כפל של $rs ב $rt במשלים ל -2 , כך שרק 32 הסיביות הנמוכות של המכפלה נשמרות באוגר $rd. (hi lo לא מכילים את המכפלה המלאה).

פסאודו־פקודות שימושיות, כפי שנתמכות בסביבת עבודה MARS

אוגר $1 ($at) הוא אוגר העזר לשימוש בפסאודו־פקודות כמוסכמת תוכנה

הפעולה	שם	תחביר	תרגום mars לפקודות אסמבלי	משמעות
העתקת $rs ל־$rt	move	move $rt,$rs	addu $rt,$0,$rs	$rt=$rs
טעינת קבוע קטן (16 סיביות) ל־$rt	load immediate (קבוע קטן)	li $rt,imm(16)	addiu $rt,imm	$rt=SE(imm0-15)
טעינת קבוע גדול (32 סיביות) ל־$rt	load immediate (קבוע גדול)	li $rt,imm(32)	lui $1,imm(16-31) ori $rt,$1,imm(0-15]	$rt=imm (32)
טעינת כתובת תווית	load address	la $rt,label	lui $1,label (16-31] ori $rt ,label(0-15]	$rt=label
טעינת מילה מהזיכרון לכתובת תווית +$rs	load word (with label)	lw $rt,label($rs)	lui $1,label [16-31] addu $1,$1,$rs lw $rt ,label[0-15]($r1)	$rt=mem(label+$rs)
שמירת $rt בזיכרון בכתובת עם תווית +$rs	store word (with label)	sw $rt,label($rs)	lui $1,label [16-31] addu $1,$1,$rs sw $rt ,label[0-15]($r1)	mem(label+$rs)=$rt
קפיצה עם $rs<$rt במשלים ל־2	branch if less than[1]	blt $rs,$rt ,label	slt $1,$rs,$rt bne $1,$0,imm	If $rs<$rt than pc=pc+4+imm*4
הפיכת הסיביות באוגר $rs והכנסת ערך זה לאוגר $rt	not	nor $rt,$rs	nor $rt,$rs,$0	$\$rt = \overline{\$rs}$
שמירת הערך המוחלט של אוגר $rs באוגר $rt	abs	abs $rt,$rs	sra $1,$rs,0x1f xor $rt,$1,$rs subu $rt,$rt,$1	$rt=abs($rs)
מעבירה את שארית החלוקה של $rs בקבוע לאוגר $rt	rem[2]	rem $rt,$rs,imm	addi $1,$0,imm div $rs,$1 mfhi $rt	$rt= $rs % imm

1. הסיבה שפסאודו־פקודה זו מפורקת לשתי פקודות אמיתיות היא מימוש ה־branch בשלב שתיים בצנרת כתנאי לוגי (שווה לא שווה), ולא כתנאי אריתמטי. קיימות עוד פסאודו־פקודות רבות של קפיצה על תנאי אריתמטי הנתמכות ב־MARS, כגון ble bleu bgt bge bgtu. כמו כן, קיימת גרסה של פסאודו־פקודות של קפיצה על תנאי המבצעות השוואה בין אוגר לקבוע.
2. קיימות מספר גרסאות של rem , כגון חלוקה בין אוגרים, או remu – חלוקה לפי ללא סימן.

ב־help של סביבת העבודה MARS ניתן למצוא את הרשימה המלאה של הפסאודו־פקודות הנתמכות (בלשונית "Extended (pseudo) Instructions").

Assembler Directives

.word W1,… ,Wn	Store n 32 bit values in successive memory words
.half H1,… ,Hn	Store n 16 bit values in successive memory half words
.byte B1, … ,Bn	Store n 8 bit values in successive memory bytes
.ascii “str”	Store ASCII string in Memory (string is a line of ASCII char)
.asciiz “str”	Store ASCII string in Memory and null-terminate it
.space n	Leave an empty n-byte region of memory (for a later use)
.align n	Align the next datum on a 2^n byte boundary For example, .align 2 the next value is word boundary

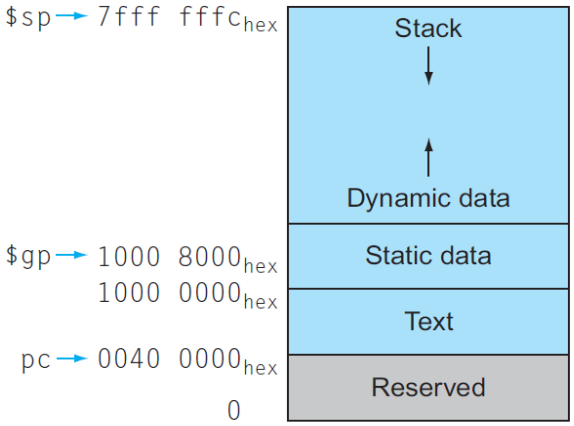
מוסכמות תוכנה בעבודה עם פרוצדורות במעבד MIPS

Name	Register number	Usage	Preserved on call?
$zero	0	The constant value 0	n.a.
$v0–$v1	2–3	Values for results and expression evaluation	no
$a0–$a3	4–7	Arguments	no
$t0–$t7	8–15	Temporaries	no
$s0–$s7	16–23	Saved	yes
$t8–$t9	24–25	More temporaries	no
$gp	28	Global pointer	yes
$sp	29	Stack pointer	yes
$fp	30	Frame pointer	yes
$ra	31	Return address	yes

FIGURE 2.14 MIPS register conventions. Register 1, called $at, is reserved for the assembler (see Section 2.12), and registers 26–27, called $k0–$k1, are reserved for the operating system. This information is also found in Column 2 of the MIPS Reference Data Card at the front of this book.

עבודת מחסנית זמן ריצה במעבד MIPS היא מוסכמת תוכנה (מערכת ההפעלה ומהדר). המוסכמה קובעת:

- מצביע ראש מחסנית הוא אוגר 29 $sp. (המחסנית היא מבנה נתונים מסוג LIFO (Last In First Out)).
- גודל איבר במחסנית הוא 4 בתים (מילה), כלומר $sp משתנה בכפולות של ארבע.
- המחסנית עובדת מכתובות גבוהות בזיכרון כלפי כתובות נמוכות.



ניהול הזיכרון הראשי על ידי מערכת ההפעלה.

גיבוי כתובת חזרה בפקודה jal מבוצע לאוגר 31 ($ra), ולכן בקריאות מקוננות (nested) יש לבצע גיבוי של אוגר 31 למחסנית.

לדוגמה:

```
addi $sp,$sp,-4  #push $ra
sw $ra,0($sp)

jal  nested_proc

lw $ra,0($sp)    #pop $ra
addi $sp,$sp,4
```

בתכנות פרוצדורלי יש להקפיד על עבודת מחסנית מסודרת. אי־שמירה על סדר עלול להביא לתופעות של גלישה (overflow) או חמיקה (underflow) במחסנית. (לדוגמה, סיום פרוצדורה הוא בנקודה אחת בפקודה jr $ra.)

פרוצדורות שירות של מערכת ההפעלה

Syscall היא פקודת אסמבלי (מכונה) של מעבד ה־MIPS הגורמת לתוכנית להפסיק על ידי חריגה[I] (exception) ולעבור לקטע קוד בגרעין של מערכת הפעלה, בשם exception handler, ומשם אנחנו מסתעפים לפי הערך של אוגר $v0 לבצע את ה־syscall המתאים.

מערכת ההפעלה מספקת פרוצדורות שירות (syscall) בתחומי ניהול תהליכים, ניהול זיכרון, ניהול קבצים ועוד. הטבלה שלהלן מסכמת את **שירותי הקלט־פלט הבסיסיים** השימושיים בקורס.

שירות \ Service	$v0	ארגומנטים \ Arguments	מחזירה \ Result
Print Integer	1	$a0 = integer value to print	-
Print String	4	$a0 = address of null-terminated	-
Read Integer	5		$v0 = integer read
Read String	8	$a0 = address of input buffer $a1 = maximum number of characters to read	המחרוזת נקלטת למקום שהוקצה מראש בסגמנט הנתונים
Exit Program	10	פרוצדורת שירות לסיום תהליך	-
Print Char	11	$a0 = character to print (low order byte)	-
Read Char	12		$v0 = character read

אופן ההפעלה: יש לעדכן את $v0 למספר פרוצדורת השירות המתאימה, במידת הצורך יש לעדכן את הארגומנטים הרלוונטיים לפרוצדורת השירות (סדר העדכון אינו חשוב).

לאחר העדכונים יש לקרוא לפרוצדורת השירות באמצעות הפקודה syscall.

חשוב להקפיד בפרוצדורות הקלט 5 ו־12. לאחר החזרה לתוכנית יש לבצע גיבוי של $v0 (שימושי להמשך).

טבלת קודי אסקי

ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter
32	space	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	DEL

הבהרות ודגשים בעבודה עם קודי אסקי ניתן למצוא בפרק ד, סעיף ד.8 במדריך.

קודי האסקי בטווח 0–31 הם תווים מיוחדים שמערכת ההפעלה מפרשת אותם כפעולה לביצוע.

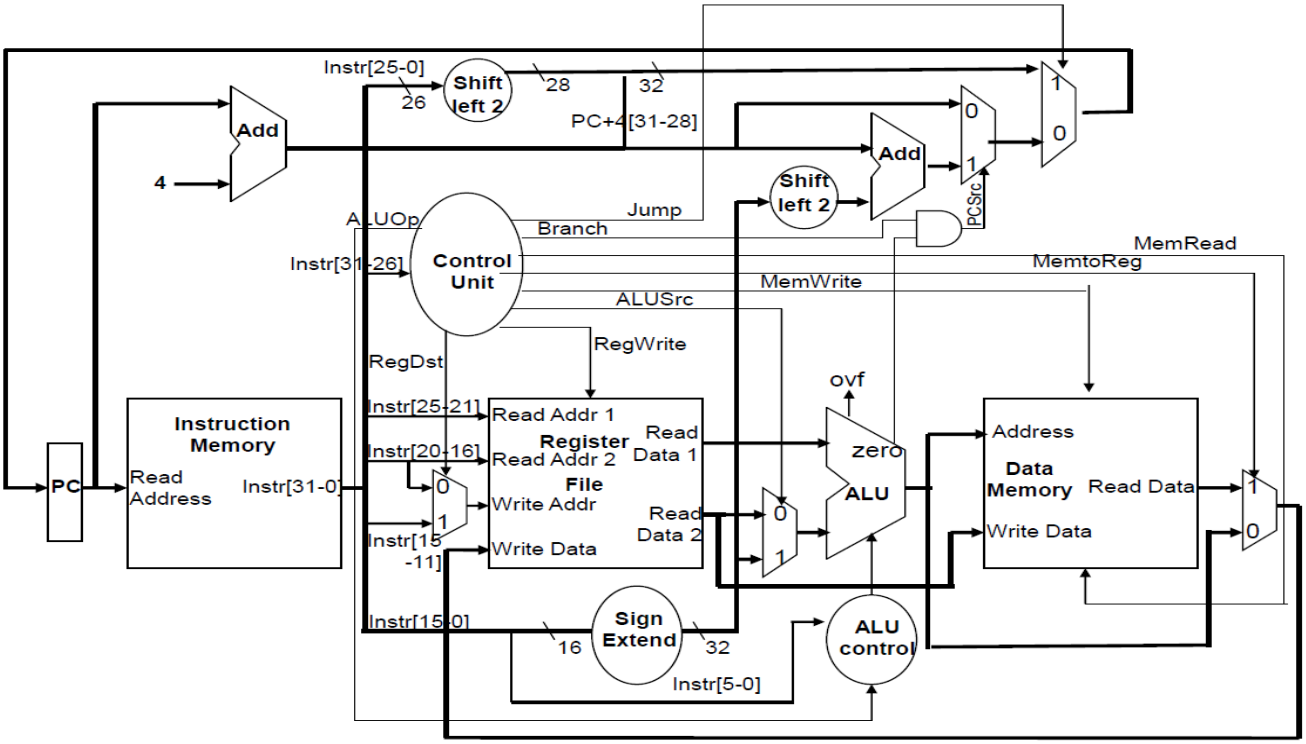
מהם הכרנו את קוד אסקי 0 המציין null terminate, ואת קוד אסקי 10 (0xa) המציין ירידת שורה ומסומן כ־'\n'.

תרשימים שימושיים במעבד חד־מחזורי

1. תרשים 4.18 קווי הבקרה הראשית, כולל תוספת הפקודות addi ו־J וקידודי הפקודות

Instruction	Opcode	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0	Jump
R-Type	0	1	0	0	1	0	0	0	1	0	0
lw	35	0	1	1	1	1	0	0	0	0	0
sw	43	x	1	x	0	0	1	0	0	0	0
beq	4	x	0	x	0	0	0	1	0	1	0
addi	8	0	1	0	1	0	0	0	0	0	0
J	2	x	x	x	0	0	0	X	X	X	1

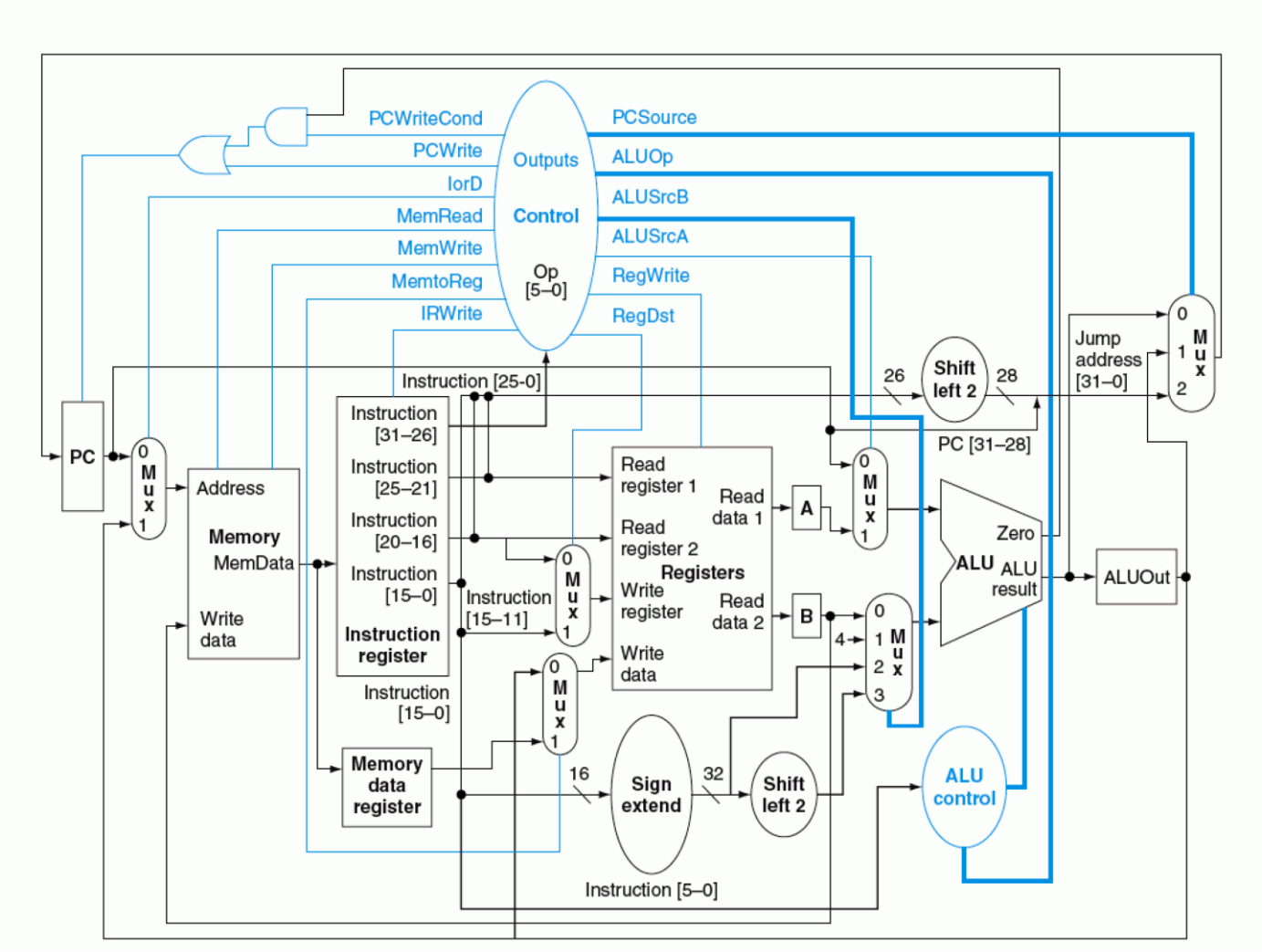
2. תרשים מעבד חד־מחזורי (בדומה לתרשים 4.24)



3) תרשים 4.12 קווי הבקרה המשנית (בקרת ה־ALU)

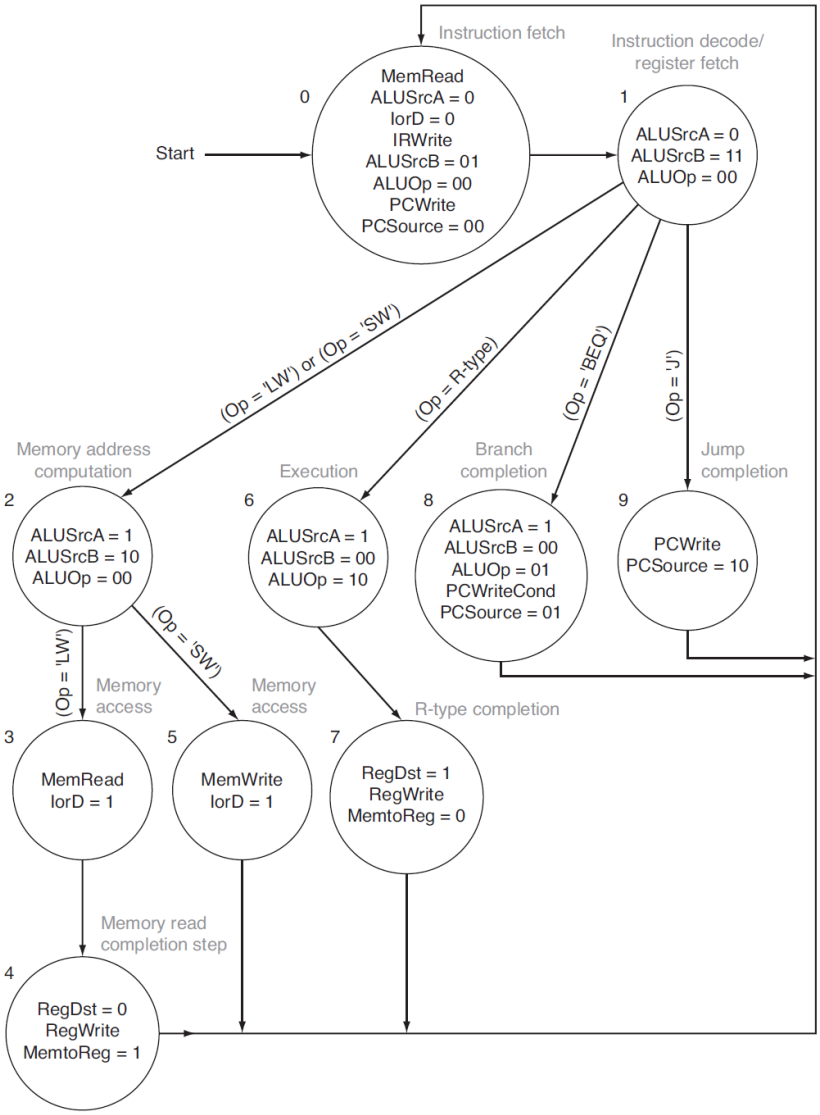
Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

נתיב הנתונים מעבד רב מחזורי.



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

מכונת מצבים מעבד רב מחזורי



תרשימים שימושיים בצנרת

Instruction	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

FIGURE 4.49 The values of the control lines are the same as in Figure 4.18, but they have been shuffled into three groups corresponding to the last three pipeline stages.

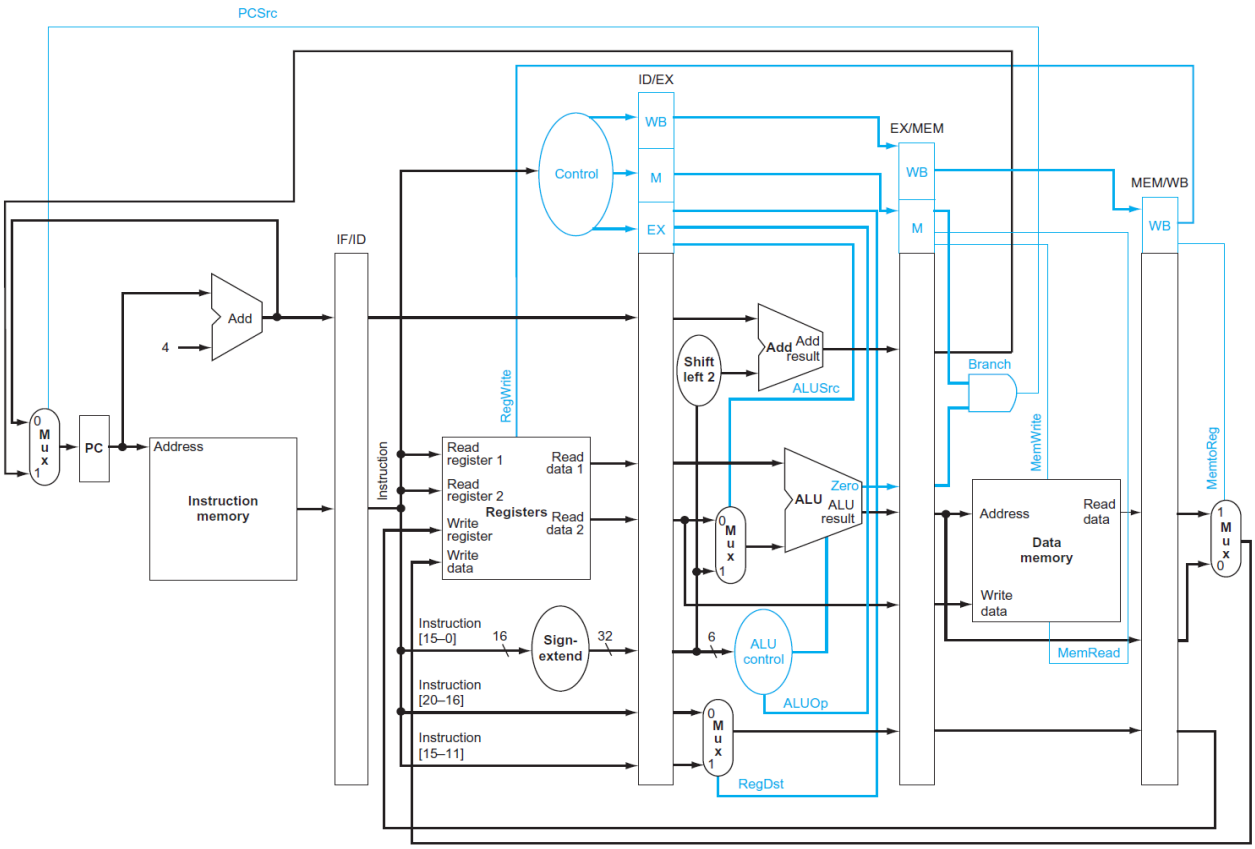


FIGURE 4.51 The pipelined datapath of Figure 4.46, with the control signals connected to the control portions of the pipeline registers. The control values for the last three stages are created during the instruction decode stage and then placed in the ID/EX pipeline register. The control lines for each pipe stage are used, and remaining control lines are then passed to the next pipeline stage.

בקרת ה־ALU מתנהגת בדיוק כמו בחד־מחזורי, ראו תרשים 4.12 במעבד חד מחזורי.

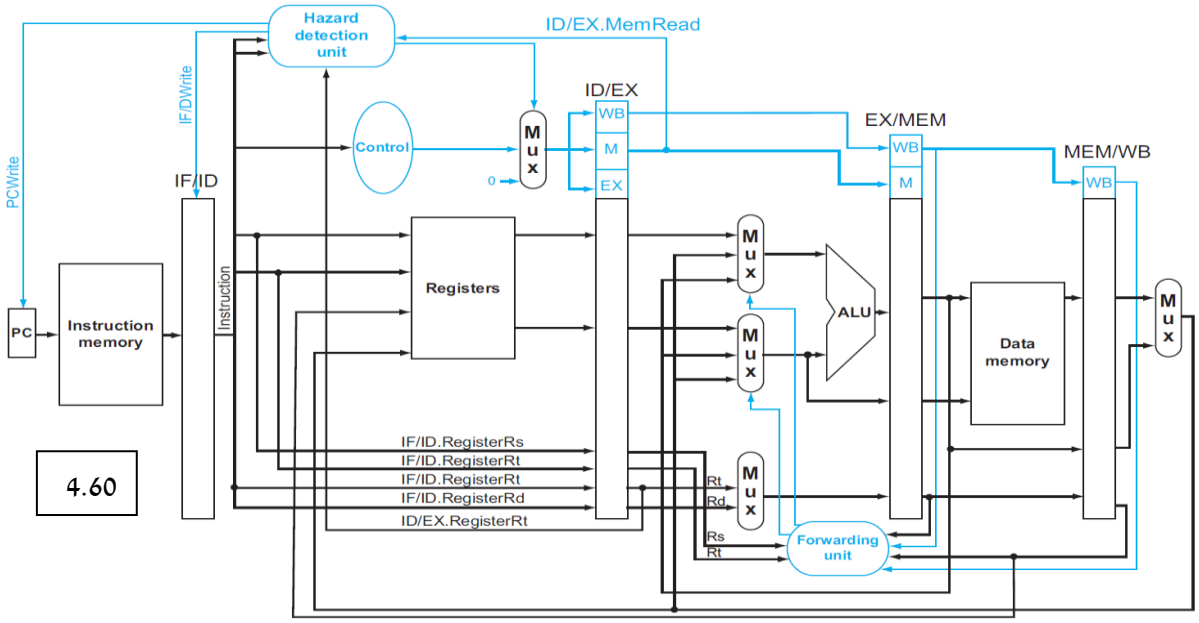
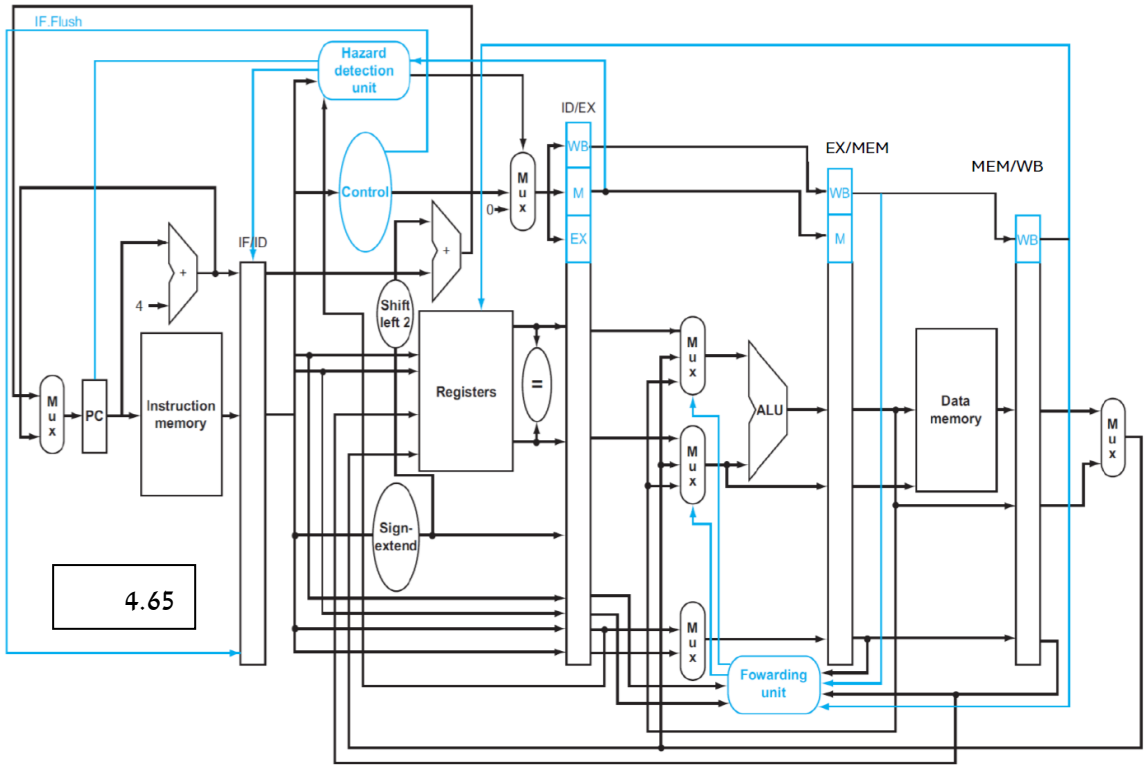


FIGURE 4.60 Pipelined control overview, showing the two multiplexors for forwarding, the hazard detection unit, and the forwarding unit. Although the ID and EX stages have been simplified—the sign-extended immediate and branch logic are missing—

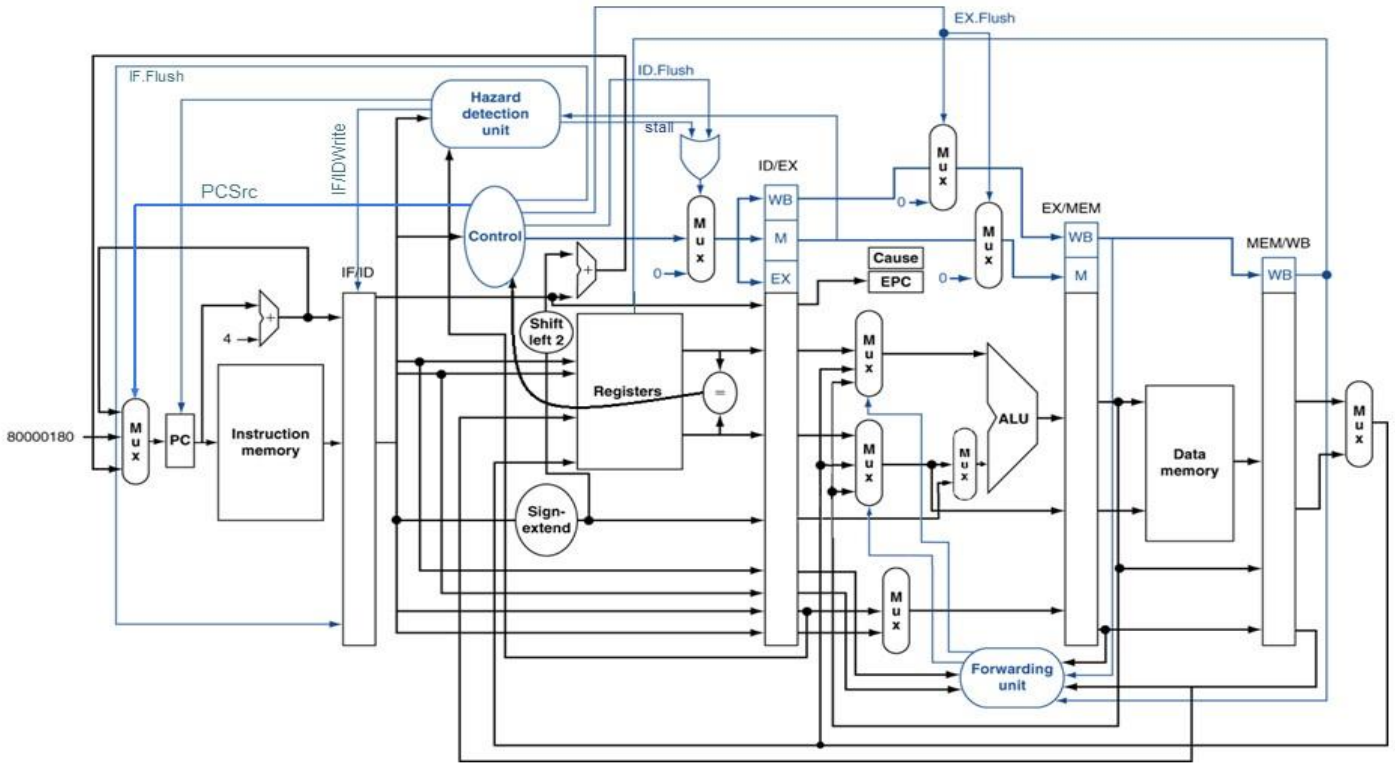


תרשים 4.60: מדגיש את סיכוני הנתונים. החיווטים של מרבבים Forward A, Forward B מוצגים בעמוד הבא. נתיב הנתונים המוצג הוא חלקי ואין בתרשים התייחסות לערך המיידי ובפרט לא מופיע המרבב של AluSrc ואין בתרשים התייחסות למימוש branch. נגדיר שבזיהוי של יחידת ה HDU ה MUX מקבל 1 וכניסה 1 מעבירה אפסים **לכל קווי הבקרה** באוגר הצנרת ID/EX ליצירת הבועה (Bubble).

תרשים 4.65: מוסיף את הקדמת ה"branch לשלב ה"ID. גם כאן נתיב הנתונים המוצג הוא חלקי.

תרשים 4.66 המשופר

כולל התייחסות לערך מיידי ולחיווט המרבבים



1. המקרים בהעברה קדימה ואופן חיווט המרבבים ForeardA, ForwardB.

באופן מקוצר נרשום את המקרים לבדיקה של יחידת העברה קדימה 1a,1b,2a,2b.

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs → ForwardA=2 (10bin)
1b. EX/MEM.RegisterRd = ID/EX.RegisterRt → ForwardB=2 (10bin)
2a. MEM/WB.RegisterRd = ID/EX.RegisterRs → ForwardA=1 (01bin)
2b. MEM/WB.RegisterRd = ID/EX.RegisterRt → ForwardB=1 (01bin)

נסכם את המקרים:

א. אם אין זיהוי של יחידת העברה קדימה מרבבים אלו יקבלו את כניסה אפס.
ב. אם יש זיהוי 1 (1a 1b) אז המרבב המתאים יחווט לכניסה 2 המחוברת ל EX/MEM.ALUResult
ג. אם יש זיהוי 2 (2a,2b) אז המרבב במתאים יחווט לכניסה 1 המחוברת ליציאה של המרבב memtoreg.

2. כניסה 1 במרבבים של ID.Flush, Ex.Flush מעבירה אפסים **לכל קווי הבקרה** באוגר הצנרת המתאים ליצירת בועה (Bubble)., ואילו כניסה 0 של אותם מרבבים מעבירה את ערכי קווי הבקרה, בהתאמה.
3. באותו אופן נניח שמצב IF.Flush=1, יגרום לאיפוס קידוד הפקודה הנמצאת בשלב IF. ולמעשה יהפוך אותה לפקודת nop (sll $0,$0,0), פקודה שכל 32 הסיביות בקידוד הן 0.
4. בכל התרשימים המכילים HDU, הקווים IF/ID.Write ו־PC.Write מקבלים 1 (enable) לציון כתיבה בסוף שעון, ורק במקרה של זיהוי load use קווים אלו יקבלו 0 (disable) ולא תתבצע כתיבה בסוף שעון.
5. ניתן להניח שהערך הנכנס ל־EPC הוא הערך של PC+4 של הפקודה הנמצאת בשלב זה. (במידת הצורך קטע הקוד המטפל בפסיקות ידע להפחית 4.)

נתונים רלוונטיים להיררכיות זיכרון

עקרון המקומיות בזמן (temporal locality) – אם ניגשנו לנתון בזיכרון, סביר שניגש אליו שוב בזמן הקרוב. עיקרון זה בא לידי ביטוי בתכנות באמצעות שימוש בלולאות, ועבור נתונים – באמצעות עדכון חוזר של נתון מסוים בזיכרון.

עקרון המקומיות במרחב (spatial locality) – אם ניגשנו לכתובת כלשהי בזיכרון, סביר שניגש בקרוב לכתובות הסמוכות לכתובת זו. עיקרון זה בא לידי ביטוי בכך שהתוכנית מתקדמת באופן סדרתי על הזיכרון, הן עבור הפקודות והן עבור מבני נתונים כגון מערכים, המאופיינים אף הם בגישה סדרתית לזיכרון הנתונים.

בלוק (או שורה): יחידת המידע הבסיסית שאפשר להעביר בין רמות זיכרון. הגודל המינימלי האפשרי של בלוק הוא מילה (32 סיביות).

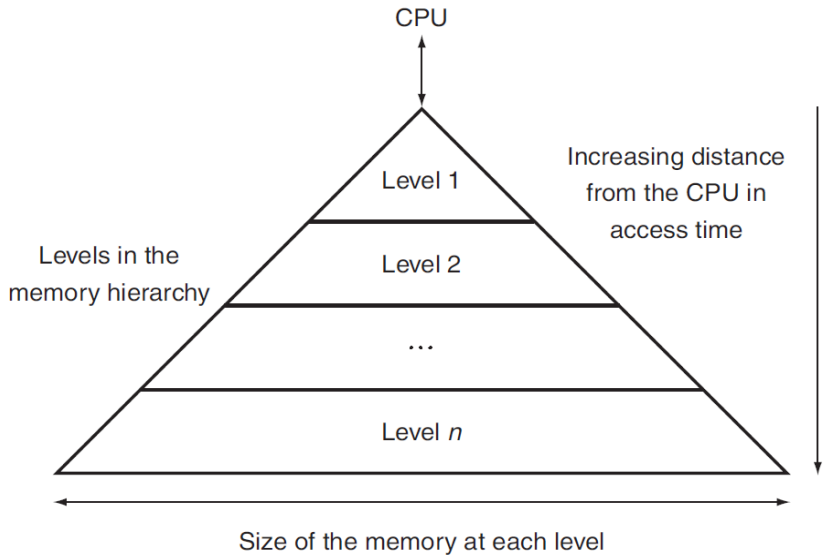


FIGURE 5.3 This diagram shows the structure of a memory hierarchy: as the distance from the processor increases, so does the size. This structure, with the appropriate operating mechanisms, allows the processor to have an access time that is determined primarily by level 1 of the hierarchy and yet have a memory as large as level *n*. Maintaining this illusion is the subject of this chapter. Although a local disk or flash memory is normally the bottom of the hierarchy, some systems use tape or a file server over a local area network as the next levels of the hierarchy.

פגיעה (hit): אם בחיפוש בלוק במטמון הבלוק נמצא במטמון, נקרא למצב זה פגיעה.
החטאה (miss): אם בחיפוש בלוק במטמון הבלוק לא נמצא, נקרא למצב זה החטאה.
יחס הפגיעה (hit rate): היחס בין כלל הפגיעות ובין המספר הכולל של הגישות למטמון.
שיעור ההחטאה (miss rate): 1-hit rate.
זמן הפגיעה (hit time): זמן הגישה לבלוק במטמון (כולל הזמן שנדרש כדי להחליט אם הבלוק נמצא או לא והעלאת הבלוק לרמה הזיכרון שמעל).
קנס ההחטאה או **זמן ההחטאה (miss penalty):** הזמן הדרוש להחלפת בלוק ברמת מטמון גבוהה בבלוק המתאים מן הרמה הנמוכה, יחד עם הזמן הנדרש כדי להעביר אותו למעבד. משך זמן הפגיעה קצר בהרבה ממשך זמן הגישה לרמה נמוכה.
הערה: בעבודה עם כמה רמות מטמון, הקנס על החטאה הוא "קנס מצטבר".

תוספת ה־CPI מאירועי miss היא המכפלה miss penalty X miss rate.

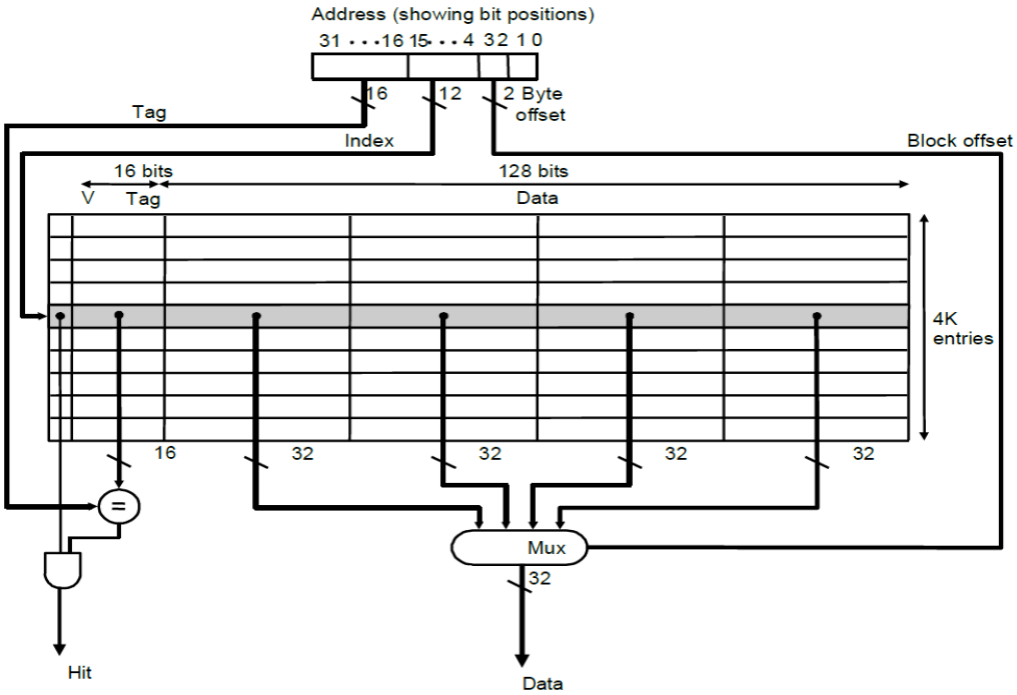
CPI=BaseCPI+ miss penalty x miss rate

בעבודה עם כמה רמות יש לבצע שקלול. לדוגמה בשתי רמות:

CPI = BaseCPI + (L1 Miss Rate/Instruction* L1 Miss Penalty) + (L1 Miss Rate * L2 Miss Rate/Instruction * L2 Miss Penalty)

מבנה מטמון במיפוי ישיר (גודל בלוק 2^m מילים):

המיפוי הוא תמיד של כתובת בזיכרון הראשי, ולכן מספר הסיביות במיפוי יגדיר את גודל הזיכרון הראשי.



- **שדה ה־index (אינדקס):** שדה זה מגדיר **חד־ערכית** את מספר השורה במטמון. **ולכן שם המיפוי הוא מיפוי ישיר**. אם יש n סיביות בשדה זה, אז יש 2^n שורות (בלוקים) במטמון. בדוגמה יש 12 סיביות בשדה האינדקס, כלומר 4,096 בלוקים/שורות.
- **שדה ה־block offset (word offset):** מגדיר את המילה בתוך הבלוק. אם יש m סיביות בשדה זה, אז יש 2^m מילים בבלוק. בתרשים ניתן לראות שיש 2 סיביות בשדה זה, כלומר יש ארבע מילים בבלוק. הסיביות של שדה זה הן בוררים של מרבב 4 ל־1ב הבוחר את המילה המתאימה בבלוק במיפוי לזיכרון הראשי. כעת גודל המטמון יהיה 2^{m+n} מילים או 2^{m+n+2} בתים. הצורה של המטמון מבחינת המידע תהיה 2^n שורות, ובכל שורה 2^m מילים או 2^{m+2} בתים.
- **שדה ה־tag (תגית):** שדה זה הוא הסיביות הגבוהות במיפוי. בצירוף הסיביות הנמוכות נקבל את הכתובת הממופה מהזיכרון הראשי באופן חד־ערכי. שדה ה־tag נשמר בזיכרון המטמון כדי לזהות באופן חד־ערכי את הכתובת הממופה מהזיכרון הראשי.
- **סיבית ה־valid (תוקף):** לכל שורה (בלוק) מוצמדת סיבית valid המציינת אם השורה במטמון שכבר מאוכלסת במידע תקף מהכתובת המתאימה בזיכרון הראשי. עם הדלקת המחשב המטמון "ריק", כלומר כל סיביות ה־valid הן 0 (לא מאוכלס).

גודל המטמון במיפוי ישיר (כולל מעטפת תגית+תוקף) בסיביות:

$2^n \times$ (גודל בלוק בסיביות + גודל תגית בסיביות + סיבית התוקף)

נדגיש שה־valid וה־tag מתייחסים לבלוק שלם רציף בזיכרון, ללא קשר לגודלו.
שימו לב שבמקרה של פגיעה (hit), עדיין צריך לרבב את המילה המתאימה מתוך הבלוק (באמצעות סיביות ה־block offset) ואז מעבירים לרמה שמעל.

Miss הנובע משדה התגית נקרא **miss_tag**. זהו מצב conflict (על אותו אינדקס מתחרות תגיות שונות).

Miss הנובע מסיבית התוקף נקרא **miss valid** (אכלוס ראשוני).

ככל שנגדיל את M כך תשתפר המקומיות במרחב, אבל תהיה גם השפעה נגדית הן על miss penalty והן על ה־conflict (miss tag).

שדה גודל בלוק	מילים	בתים	סיביות
M=0	1	4	32
M=1	2	8	64
M=2	4	16	128
M=3	8	32	256
M=4	16	64	512
M=5	32	128	1024
M=6	64	256	2048
M=7	128	512	4096

חזקת 10 ב־SI	ערך ב־SI	סימון ב־SI	שם ב־SI		חזקת 2 IEC	ערך ב־IEC	סימון ב־IEC	שם ב־IEC
10^3	1 000	K	Kilo		2^{10}	1,024	Ki	Kibi
10^6	1 000 000	M	Mega		2^{20}	1,048,576	Mi	Mebi
10^9	1 000 000 000	G	Giga		2^{30}	1,073,741,824	Gi	Gibi
10^{12}	1 000 000 000 000	T	Terra		2^{40}	1,099,511,627,776	Ti	Tebi

הצגת גודלי הזיכרון בחזקות של 2 בבתים.

תהליך המיפוי מכתובת בזיכרון הראשי למציאת המיקום במטמון על פי נתוניו (m, n)

להלן המתכון לאופן המיפוי הכללי מבחינת הסתכלות על השדות:

tag, index, block offset, byte offset

- קודם כול שימו לב שיש שאלות בתרגילים או במבחנים המתייחסות לכתובת בבתים במילים, בהתאמה. אם הכתובת ניתנת כבר במילים, פשוט נתעלם מה־byte offset.

דרך ראשונה: להצגה היא באמצעות רישום בינרי של הכתובת הממופה מהזיכרון הראשי, וכך להגיע לחלוקה המתאימה לשדות.

דרך שנייה:

שלב ראשון: "להעלים" את ההיסטים (offset), כלומר לחלק בגודל הבלוק מבלי להתייחס לשארית (הנמצאת בהיסטים) – החלוקה במילים או בבתים (כלומר 2^m או 2^{m+2}) לפי נתוני השאלה נותנת לנו מספר הנקרא block address (לפעמים בנתוני שאלה של גודל בלוק שווה מילה ובכתובות המופיעות במילים למעשה נבצע חילוק ב־1, כלומר מתעלמים מראש מההיסטים וניתן לדלג על שלב זה).

כעת למעשה נשארנו עם מספר הבלוק בזיכרון הראשי (block address), כלומר עם סיביות האינדקס וה־tag במיפוי.

שלב שני: כדי להפריד בין האינדקס ל־tag נחלק את block address במספר השורות/בלוקים (2^n). מטמון השארית ייתן את האינדקס והתוצאה תיתן את ה־tag.

(ה־block addres הוא צירוף של tag+index במיפוי, ומתאר את מספר הבלוקים בזיכרון הראשי. ומכאן נובעת ההפרדה index, tag דרך mod ו־div.)

צמצום החטאות באמצעות אסוציאטיביות:

לנוסחאות חישוב גודל המטמון יש להוסיף את פקטור K (דרגת האסוציאטיביות) שלא מופיע במיפוי של זיכרון המטמון (המראה מיפוי של K=1, כלומר מיפוי ישיר). המקרה הכללי הוא שבמטמון יש $\{2^n \times K\}$ בלוקים, ולכן המספר הכללי (כולל tag+valid) של סיביות במטמון הוא:

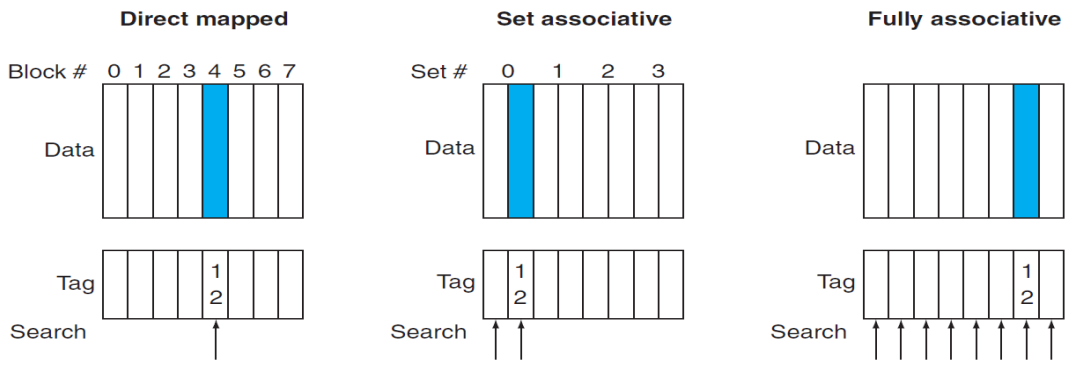
{מספר בלוקים}×(כמות מידע בבלוק+ תגית+תוקף) בסיביות} = $\{K \times 2^n\} \times \{2^{m+5}+tag+valid\}$

במידה ונרצה רק את גודל הנתונים אותם ניתן לאפסן במטמון:

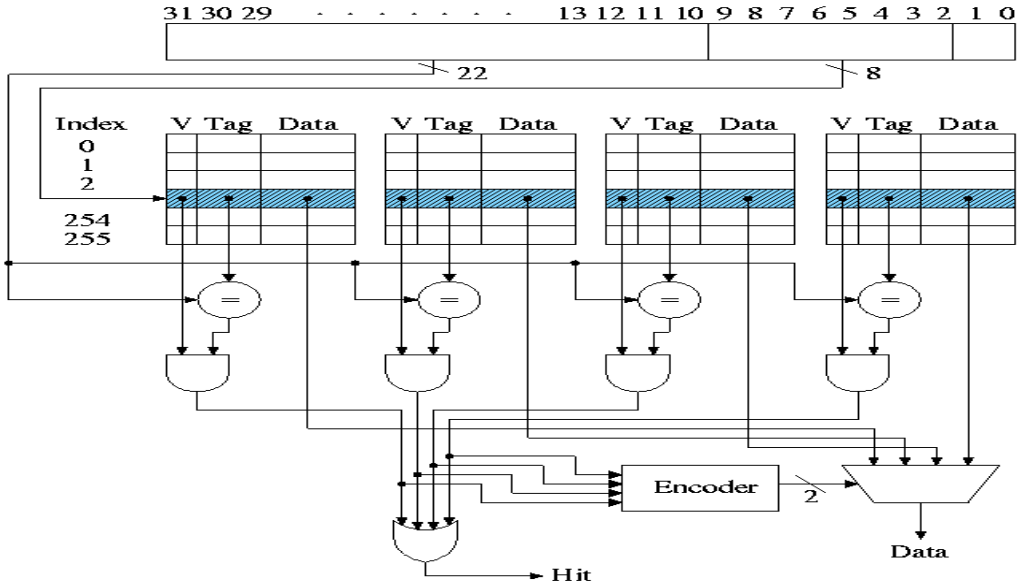
$\{K \times 2^n\} \times \{2^{m+5}\}$ בסיביות [bit]

$\{K \times 2^n\} \times \{2^{m+2}\}$ בבתים [byte]

$\{K \times 2^n\} \times \{2^m\}$ במילים [word]



בזיכרון קבוצתי־אסוציאטיבי (K-way set-associative cache) מהמיפוי מגיעים לאינדקס של הסט \שורה המתאימה, ובתוך הסט\שורה עצמה יש להשוות את כל התגיות לתגית המבוקשת.



המטמון הזה הוא 4-way set associative cache. כמות הנתונים במטמון היא:

2^8(שורות)*4 (set)*32[bit] (גודל בלוק מילה)=32,768[bit]=4KByte

כמות הנתונים במטמון כולל המעטפת (valid tag) היא:

2^8(שורות)*4 (set)*(32+22+1)[bit]= 56,320[bit]

במטמון אסוציאטיבי מלא (fully associative), כל בלוק יכול להיכנס לכל כניסה של זיכרון המטמון. כאשר משתמשים במיפוי כזה, כדי למצוא בלוק יש לבדוק את כל הכניסות, וכדי לייעל את תהליך הבדיקה מבצעים את הבדיקות בו־זמנית, באמצעות משווה (comparator) המחובר לכל כניסה. שימו לב שבמצב זה **לא קיים כלל שדה אינדקס (n = 0)**, וניתן לראות את כל זיכרון המטמון כסט\שורה אחת.

הבחירה בבלוק שיוחלף: הסכֵמה הנפוצה ביותר לבחירת בלוק היא **LRU** (Least Recently Used). בשיטה זו מחליפים את הבלוק שלא היה בשימוש הזמן רב ביותר.

MIPS Reference Data

①

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31–26	25–21	20–16	15–11	10–6	5–0
I	opcode	rs	rt	immediate		
	31–26	25–21	20–16	15–0		
J	opcode	address				
	31–26	25–0				

ARITHMETIC CORE INSTRUCTION SET

②

NAME, MNEMONIC		FOR-MAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.*x*.s*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	c.*x*.d*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/*y*
* (*x* is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)					
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0 /--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0 /--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		10 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >>> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31–26	25–21	20–16	15–11	10–6	5–0
FI	opcode	fmt	ft	immediate		
	31–26	25–21	20–16	15–0		

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

③

OPCODES, BASE CONVERSION, ASCII SYMBOLS

MIPS opcode (31:26)	(1) MIPS funct (5:0)	(2) MIPS funct (5:0)	Binary	Decimal	Hexadecimal	ASCII Character	Decimal	Hexadecimal	ASCII Character
(1)	sll	add.*f*	00 0000	0	0	NUL	64	40	@
		sub.*f*	00 0001	1	1	SOH	65	41	A
j	srl	mul.*f*	00 0010	2	2	STX	66	42	B
jal	sra	div.*f*	00 0011	3	3	ETX	67	43	C
beq	sllv	sqrt.*f*	00 0100	4	4	EOT	68	44	D
bne		abs.*f*	00 0101	5	5	ENQ	69	45	E
blez	srlv	mov.*f*	00 0110	6	6	ACK	70	46	F
bgtz	srav	neg.*f*	00 0111	7	7	BEL	71	47	G
addi	jr		00 1000	8	8	BS	72	48	H
addiu	jalr		00 1001	9	9	HT	73	49	I
slti	movz		00 1010	10	a	LF	74	4a	J
sltiu	movn		00 1011	11	b	VT	75	4b	K
andi	syscall	round.w.*f*	00 1100	12	c	FF	76	4c	L
ori	break	trunc.w.*f*	00 1101	13	d	CR	77	4d	M
xori		ceil.w.*f*	00 1110	14	e	SO	78	4e	N
lui	sync	floor.w.*f*	00 1111	15	f	SI	79	4f	O
	mfhi		01 0000	16	10	DLE	80	50	P
(2)	mthi		01 0001	17	11	DC1	81	51	Q
	mflo	movz.*f*	01 0010	18	12	DC2	82	52	R
	mtlo	movn.*f*	01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
	mult		01 1000	24	18	CAN	88	58	X
	multu		01 1001	25	19	EM	89	59	Y
	div		01 1010	26	1a	SUB	90	5a	Z
	divu		01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	]
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	_
lb	add	cvt.s.*f*	10 0000	32	20	Space	96	60	`
lh	addu	cvt.d.*f*	10 0001	33	21	!	97	61	a
lwl	sub		10 0010	34	22	"	98	62	b
lw	subu		10 0011	35	23	#	99	63	c
lbu	and	cvt.w.*f*	10 0100	36	24	$	100	64	d
lhu	or		10 0101	37	25	%	101	65	e
lwr	xor		10 0110	38	26	&	102	66	f
	nor		10 0111	39	27	'	103	67	g
sb			10 1000	40	28	(	104	68	h
sh			10 1001	41	29	)	105	69	i
swl	slt		10 1010	42	2a	*	106	6a	j
sw	sltu		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l
			10 1101	45	2d	-	109	6d	m
swr			10 1110	46	2e	.	110	6e	n
cache			10 1111	47	2f	/	111	6f	o
ll	tge	c.f.*f*	11 0000	48	30	0	112	70	p
lwc1	tgeu	c.un.*f*	11 0001	49	31	1	113	71	q
lwc2	tlt	c.eq.*f*	11 0010	50	32	2	114	72	r
pref	tltu	c.ueq.*f*	11 0011	51	33	3	115	73	s
	teq	c.olt.*f*	11 0100	52	34	4	116	74	t
ldc1		c.ult.*f*	11 0101	53	35	5	117	75	u
ldc2	tne	c.ole.*f*	11 0110	54	36	6	118	76	v
		c.ule.*f*	11 0111	55	37	7	119	77	w
sc		c.sf.*f*	11 1000	56	38	8	120	78	x
swc1		c.ngle.*f*	11 1001	57	39	9	121	79	y
swc2		c.seq.*f*	11 1010	58	3a	:	122	7a	z
		c.ngl.*f*	11 1011	59	3b	;	123	7b	{
		c.lt.*f*	11 1100	60	3c	<	124	7c	|
sdc1		c.nge.*f*	11 1101	61	3d	=	125	7d	}
sdc2		c.le.*f*	11 1110	62	3e	>	126	7e	~
		c.ngt.*f*	11 1111	63	3f	?	127	7f	DEL

(1) opcode(31:26) == 0

(2) opcode(31:26) == 17_{ten} (11_{hex}); if fmt(25:21)==16_{ten} (10_{hex}) *f* = s (single); if fmt(25:21)==17_{ten} (11_{hex}) *f* = d (double)

IEEE 754 FLOATING-POINT STANDARD

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

where Single Precision Bias = 127,
Double Precision Bias = 1023.

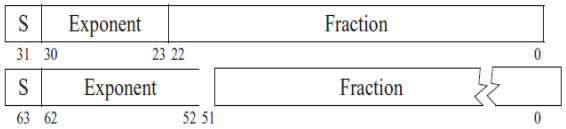
④

IEEE 754 Symbols

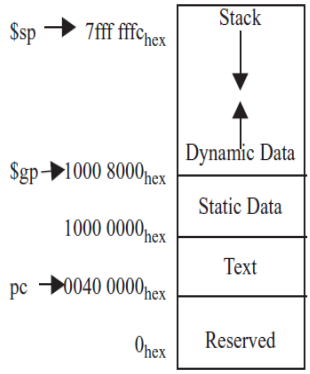
Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

S.P. MAX = 255, D.P. MAX = 2047

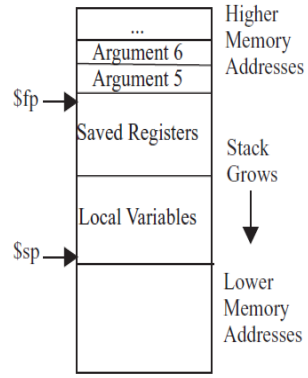
IEEE Single Precision and Double Precision Formats:



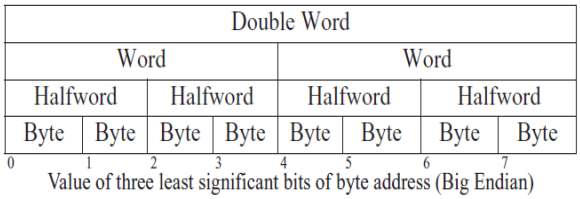
MEMORY ALLOCATION



STACK FRAME

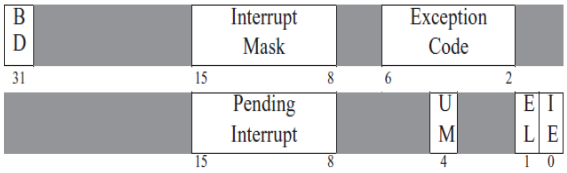


DATA ALIGNMENT



Value of three least significant bits of byte address (Big Endian)

EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS



BD = Branch Delay, UM = User Mode, EL = Exception Level, IE =Interrupt Enable

EXCEPTION CODES

Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	Syscall Exception	15	FPE	Floating Point Exception

SIZE PREFIXES (10^x for Disk, Communication; 2^x for Memory)

SIZE	PREFIX	SIZE	PREFIX	SIZE	PREFIX	SIZE	PREFIX
10^3, 2^{10}	Kilo-	10^{15}, 2^{50}	Peta-	10^{-3}	milli-	10^{-15}	femto-
10^6, 2^{20}	Mega-	10^{18}, 2^{60}	Exa-	10^{-6}	micro-	10^{-18}	atto-
10^9, 2^{30}	Giga-	10^{21}, 2^{70}	Zetta-	10^{-9}	nano-	10^{-21}	zepto-
10^{12}, 2^{40}	Tera-	10^{24}, 2^{80}	Yotta-	10^{-12}	pico-	10^{-24}	yocto-

The symbol for each prefix is just its first letter, except μ is used for micro.