מבוא למחשבים
Lecture 2

סיווג מעבדים
שיטות מיעון
SRC אסמבלי

ד"ר רון שמואלי

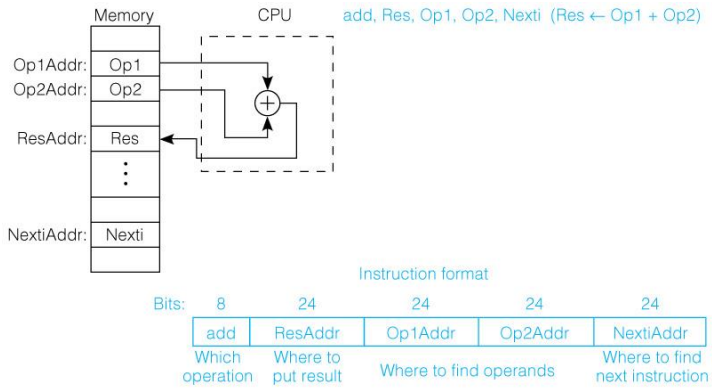
חלק נכבד מהשקפים מבוסס על הספר:

- Heuring and Jordan: "Computer System Design and Architecture", **Prentice Hall**, 2004

CPU Classification based on arithmetic instructions

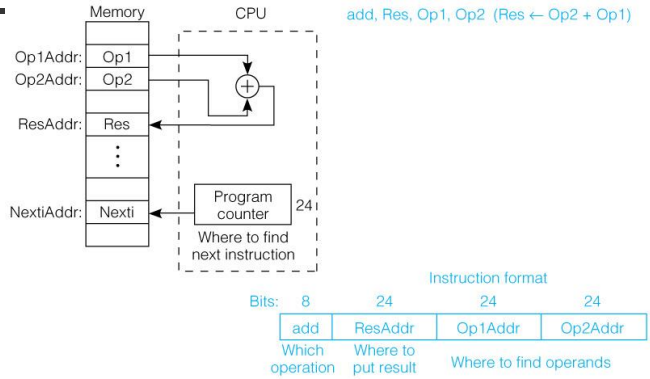
- Classification based on arithmetic instructions that have two operands and one result
- The 4-address instruction, R ← Op1 op Op2
  the address of the next instruction to specified explicitly.
- The 3, Address Instructions R ← Op1 op Op2
- The 2, Address Instructions Op2 ← Op1 op Op2
- The 1, Address Instructions Acc ← Acc op Op1 (accumulator register)

Fig. 2.3 The 4 Address Instruction



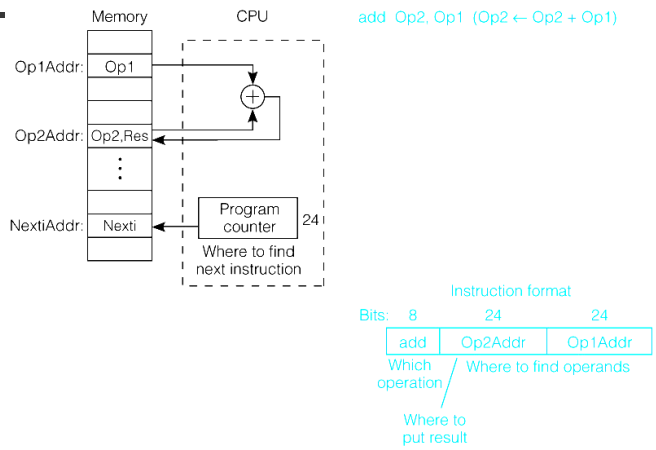
- Explicit addresses for operands, result & next instruction
- Example assumes 24-bit addresses
  - Discuss: size of instruction in bytes

Fig 2.4 The 3 Address Instruction



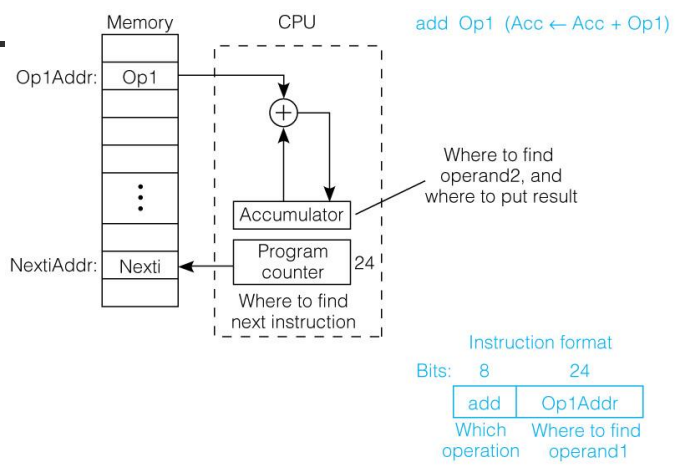
- Address of next instruction kept in a processor state register—the PC (Except for explicit Branches/Jumps)
- Rest of addresses in instruction
  - Discuss: savings in instruction word size

Fig. 2.5 The 2 Address Instruction



- Be aware of the difference between address, Op1Addr, and data stored at that address, Op1.
- Result overwrites Operand 2, Op2, with result, Res
- This format needs only 2 addresses in the instruction but there is less choice in placing data

Fig. 2.6 1 Address Instructions



- Special CPU register, the accumulator, supplies 1 operand and stores result
- One memory address used for other operand

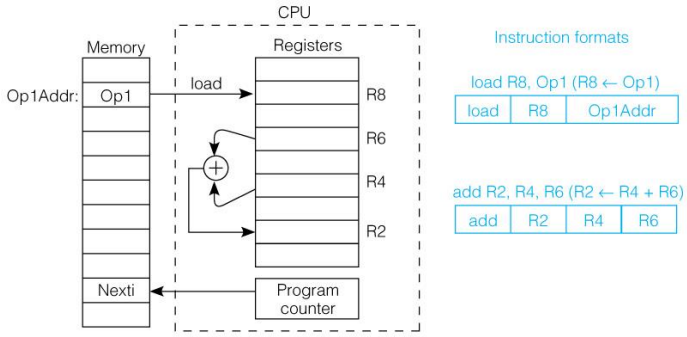
Example 2.1 Expression evaluation for 3-1 address instructions.

Evaluate a = (b+c)*d-e for 3- 2- 1- and 0-address machines.

3-Address	2-Address	Accumulator
add a,b,c	load a,b	lda b
mpy a,a,d	add a,c	add c
sub a,a,e	mpy a,d	mpy d
	sub a,e	sub e
		sta a

- # of instructions & # of addresses both vary
- Discuss as examples: size of code in each case

Fig. 2.8 General Register Machines Load/Stare Machine (RISC)



- It is the most common choice in today's general purpose computers
- *Which* register is specified by small "address" (3 to 6 bits for 8 to 64 registers)
- Load and store have one long & one short address: 1 1/2 addresses
- 2-Operand arithmetic instruction has 3 "half" addresses

Fig 2.9 Common Addressing Modes a-d
שיטות מיעון

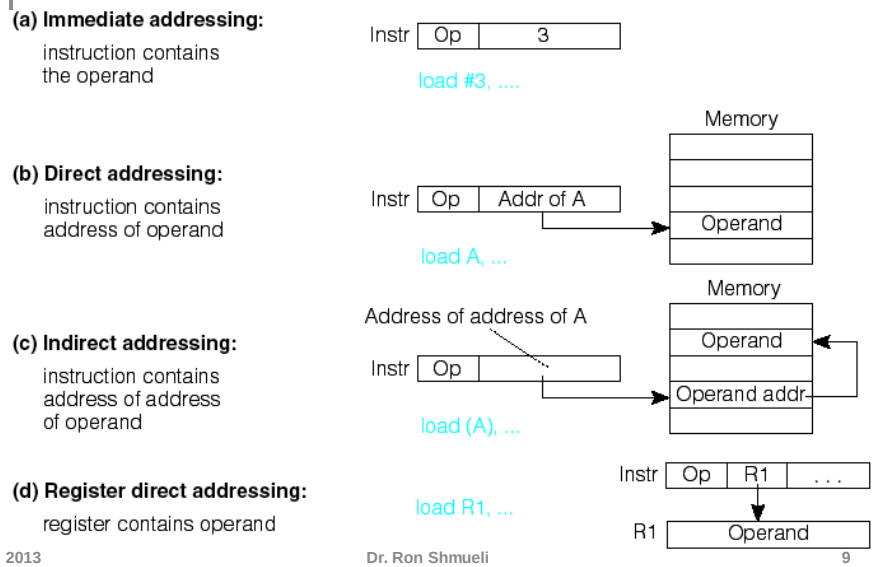
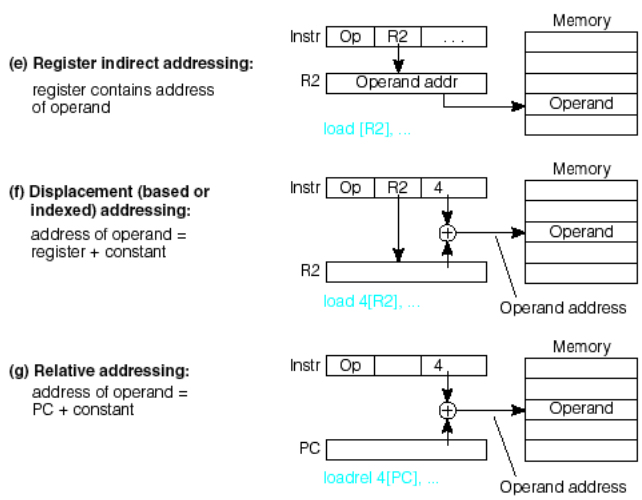


Fig 2.9 Common Addressing Modes e-g



CISC= COMPLEX INSTRUCTION SET COMPUTER

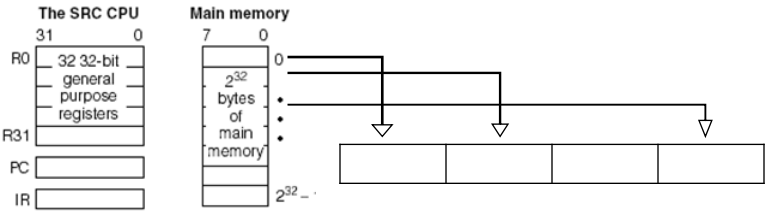
- **מאפיינים עיקריים**
  - מספר גדול של פקודות, פקודות מסובכות.
  - הידור פשוט – התאמת פקודת אסמבלי לפקודות בשפה עילית.
  - פקודות באורך משתנה –
    - **שתי כתובות זיכרון ← פקודות ארוכות.**
    - **שני אוגרים ← פקודות קצרות.**
  - שילוב של מספר שיטות מיעון באותה פקודה.
  - ביצוע פעולות על אופרנדים בזיכרון.
  - קידוד משתנה- פקודות שכיחות קיבלו קוד קצר.
  - שימוש במיקרו תכנות ביישום המעבד.
- **חסרונות**
  - תהליך FETCH ו- DECODING מורכב – מאט את המעבד.
  - תכנון מעבד מסובך – תקלות תכנון יקרות.
  - שימוש נדיר בחלק מהפקודות.

RISC=Reduced Instruction Set Computer

- הארכיטקטורה נועדה לקיצור זמן ביצוע
- **מאפיינים עיקריים:**
  - מספר קטן של פקודות
  - מספר מצומצם של שיטות מיעון
  - גישה לזיכרון רק בפקודות Load/Store.
  - כל הפעולות מבוצעות בין אוגרים
  - אורך פקודה קבוע (קל ומהיר לפענוח)
  - מאפשר - ביצוע פקודה במחזור השעון יחיד בארכיטקטורת PIPELINE.
  - חומרה מוקשחת זולה ומהירה יותר.
- **חסרונות**
  - מספר גדול של אוגרים.
  - הקידוד דורש מספר גדול יותר של פקודות.

Fig. 2.10a The SRC Simple RISC Computer

- 32 general purpose registers of 32 bits
- 32 bit program counter, PC and instruction reg., IR
- 2^{32} bytes of memory address space



R[7] means contents of register 7

M[32] means contents of memory location 32

Programmer's Model: Instruction Set Architecture (ISA)

- ISA includes
  - instruction set + RTL
  - Commands structure (Size and meaning of each filed)
  - Memory resources , and I/O
  - programmer accessible registers.
  - SRC Registers
    - IR
    - PC
    - MA
    - MD
    - Register file

opcode	Operands

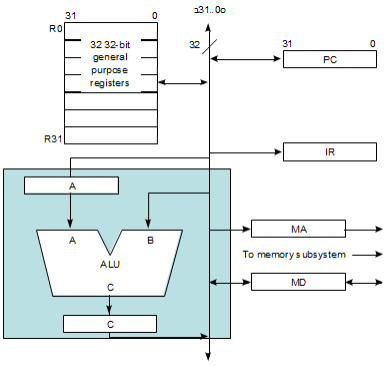
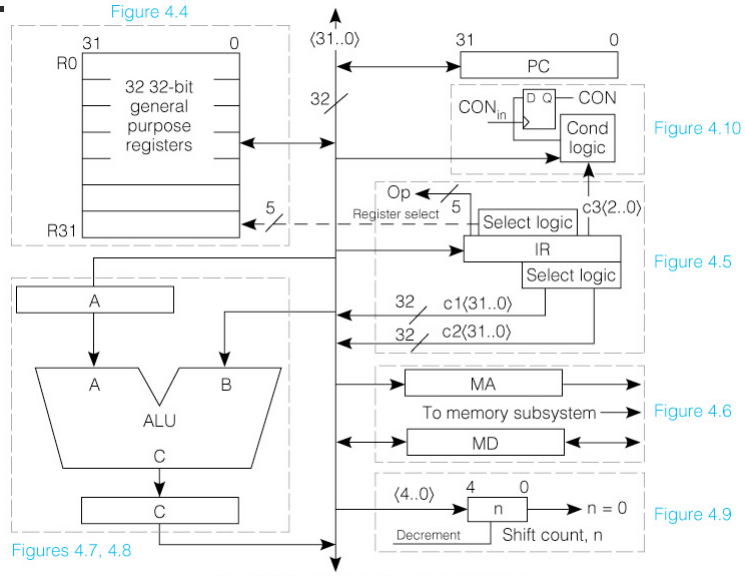
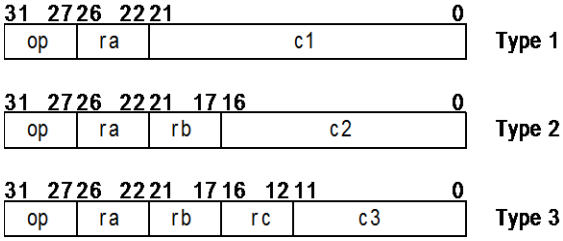


Fig. 4.3 More Complete view of Registers and Buses in 1-bus SRC



SRC Basic Instruction Formats

- There are three basic instruction format types
- The number of register specifier fields and length of the constant field vary
- Other formats result from unused fields or parts



- **Details of formats:**

Fig 2.10 SRC instructions

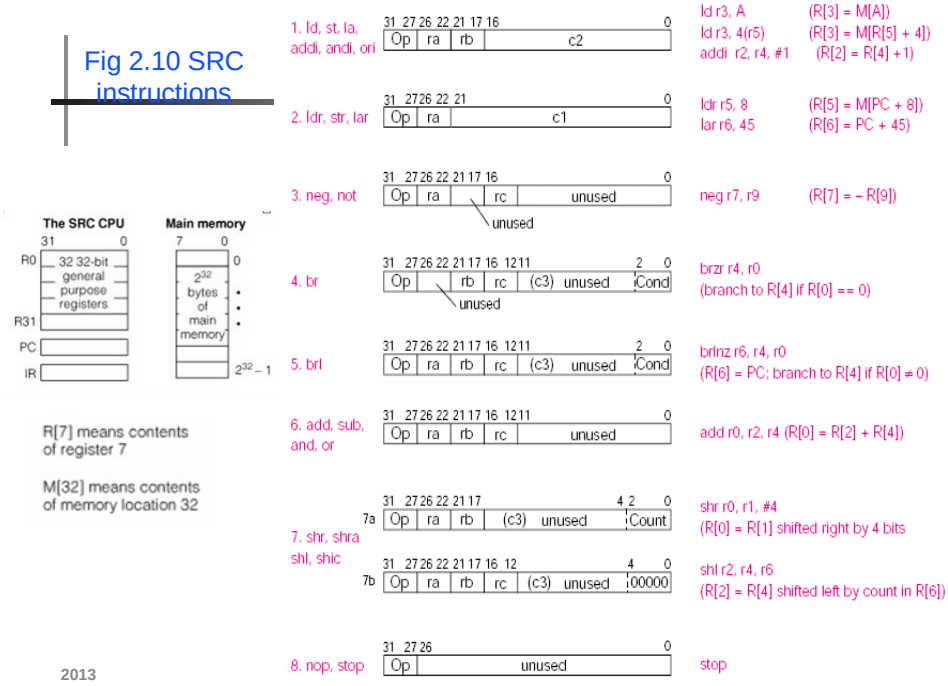
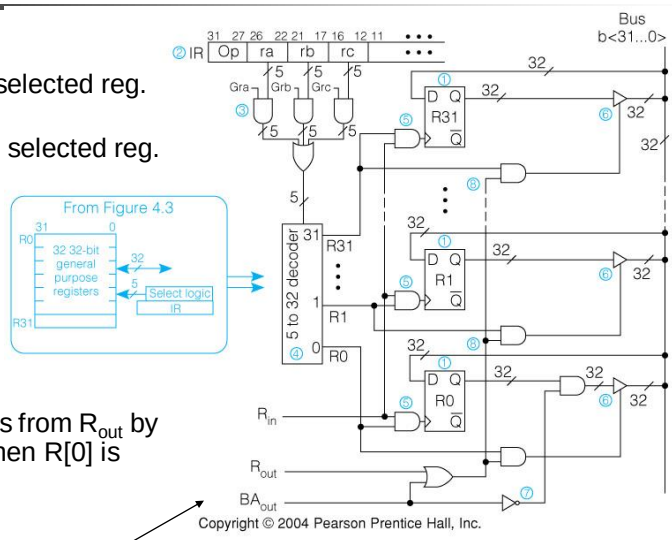


Fig. 4.4 The SRC Register File and Its Control Signals

- R_{out} gates selected reg. onto bus
- R_{in} strobed selected reg. from bus



- BA_{out} differs from R_{out} by gating 0 when R[0] is selected

BA = Base Address

Format 1

1. ld, st, la, addi, andi, ori

31 27	26 22	21 17	16 0
Op	ra	rb	c2

ld r3, A (R[3] = M[A])
ld r3, 4(r5) (R[3] = M[R[5] + 4])
addi r2, r4, #1 (R[2] = R[4] + 1)

ld - Load op =1

rb=0 → ld ra,c2 R[ra]←M[c2]
rb ≠0 → ld ra,c2(rb) R[ra]←M[c2+R[rb]]

Instruction	op	ra	rb	c 2	Meaning	Addressing Mode
ld r1, 32	1	1	0	32	R[1] ← M[32]	Direct
ld r22, 24(r4)	1	22	4	24	R[22] ← M[24+R[4]]	Displacement

la – Load Displacement Address op =5

rb=0 → la ra,c2 R[ra]←c2
rb ≠0 → ld ra,c2(rb) R[ra]←c2+R[rb]

la r7, 32	5	7	0	32	R[7] ← 32	Immediate
la r22, 24(r4)	**5**	22	4	24	R[22] ← [24+R[4]]	

(note use of la to load a constant) **rb r0 לא יכול לשמש כ**

Format 1

st - store op =3

rb=0 → st ra,c2 M[c2] ←R[ra]
rb ≠0 → st ra,c2(rb) M[c2+R[rb]]← R[ra]

Instruction	op	ra	rb	c2	Meaning	Addressing Mode
st r4, 0(r9)	3	4	9	0	M[R[9]] ← R[4]	Register indirect
st r4, 0	3	4	**0**	0	M[**0**] ← R[4]	

addi andi ori

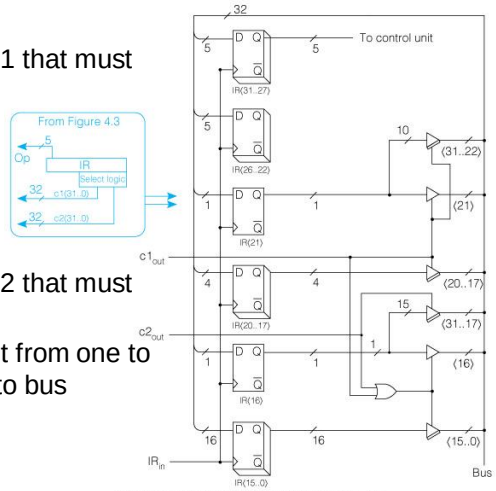
OP ra, rb, c2 R[ra]←R[rb] <op> c2

addi ra, rb, c2 addi r1, r3, 1 ;Immediate 2's complement add
andi ra, rb, c2 ;Immediate logical and
ori ra, rb, c2 ;Immediate logical or

c2 <17 bits> + R[rb] < 32 bits> **הערה: טיפול ברחבת סימן**

Fig. 4.5 Extracting c1, c2, and op from the Instruction Register

- I⟨21⟩ is the sign bit of C1 that must be extended
- I⟨16⟩ is the sign bit of C2 that must be extended
- Sign bits are fanned out from one to several bits and gated to bus



Format 2

2. ldr, str, lar

31 27	26 22	21 0
Op	ra	c1

- **ldr – load data relative (to PC)**

  ldr ra,c1 **R[ra] = M[PC+c1]**

  ldr r5, 8 (R[5] = M[PC + 8])

Instruction	op	ra	rb	c1	Meaning	Addressing Mode
ldr r12, -48	2	12	–	-48	R[12] ← M[PC -48]	Relative

- **lar – load address relative (to PC)**

  lar ra,c1 **R[ra] = PC+c1**

  lar r6, 45 (R[6] = PC + 45)

lar r3, 0	6	3	–	0	R[3] ← PC	Register (!)

- **str – store data relative (to PC)**

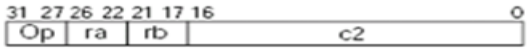
  str ra,c1 **M[PC+c1] = R[ra]**

  str r6,45 **M[PC+45] = R[6]**

דוגמא

Address	Data
100	0D
101	88
102	00
103	18

- נתונה תכולת הזכרון ב Hex – מהי הפקודה?

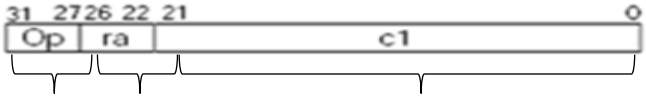


(0D880018) = 0000 1011 1000 1000 0000 0000 0001 1000

ld r22 r4 24

- נתונה הפקודה ldr r12, -48 נובת 100 מה תכולת הזיכרון ?

Address	Data
100	11
101	BF
102	FF
103	D0



00010 00110 1111111111111111010000

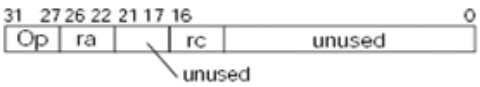
- בזמן ריצה מאיזו יובא הנתון

ldr r12, -48 R[12] = M[PC+(-48)]

56

Format 3

3. neg, not



- neg ra,rc R[ra] = - R[rc] (2's comp.)
- not ra,rc R[ra] = not(R[rc]) (1's comp.)

Format 8

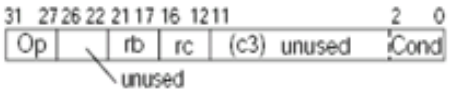
8. nop, stop

stop

- NOP = No Operation

Format 4

4. br



- משפחה של פקודות הסתעפות בפורמט הבא (opcode=8):
- **brxx rb,rc,c3<2..0>**
- **משמעות הפקודה:**

אם התנאי המופיע בשדה c3 המופעל על R[rc] מתקיים,

בצע הסתעפות לכתובת באוגר R[rb] (כלומר PC← R[rb]).

lsbs	condition	Assy language form	Example
000	never	brlnv	brlnv r6
001	always	br, brl	br r5, brl r5
010	if rc = 0	brzr, brlzr	brzr r2, r4, r5
011	if rc ≠ 0	brnz, brlnz	
100	if rc >= 0	brpl, brlpl	
101	if rc < 0	brmi, brlmi	

Branch Instructions—Example

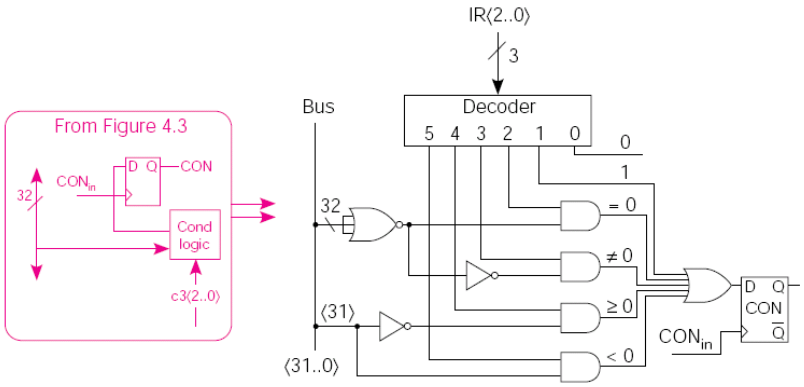
Ass'y lang.	Example instr.	Meaning	op	ra	rb	rc	c3 ⟨2..0⟩	Branch Cond'n.
br	br r4	PC ← R[4]	8	—	4	—	001	always
brzr	brzr r5,r1	if (R[1]=0) PC ← R[5]	8	—	5	1	010	zero

```
C: goto Label3

SRC:
        lar  r0,  Label3; put branch target address into tgt reg.
        br r0             ; and branch
        • • •

Label3  • • •
```

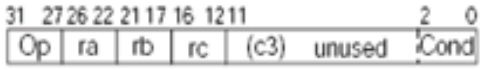
Fig 4.9 Computation of the Conditional Value CON



- **NOR gate does = 0 test of R[rc] on bus**

Format 5

5. brl



- משפחה של פקודות הסתעפות עם שמירת כתובת חזרה (opcode=9):
- **brlxx ra,rb,rc,c3<2..0>**
- **משמעות הפקודה:**

שמור את ה PC הנוכחי R[ra]←PC

אם התנאי המופיע בשדה c3 המופעל על R[rc] מתקיים,

אזי: בצע הסתעפות לכתובת באוגר R[rb] (PC← R[rb]).

lsbs	condition	Assy language form	Example
000	never	brlnv	brlnv r6
001	always	br, brl	br r5, brl r5
010	if rc = 0	brzr, brlzr	brzr r2, r4, r5
011	if rc ≠ 0	brnz, brlnz	
100	if rc >= 0	brpl, brlpl	
101	if rc < 0	brmi, brlmi	

דוגמאות

Ass'y lang.	Example instr.	Meaning	op	ra	rb	rc	c3 ⟨2..0⟩	Branch Cond'n.
brlnz	brlnz r2,r1,r0	R[2] ← PC; if (R[0]≠0) PC← R[1]	9	2	1	0	011	nonzero
brlpl	brlpl r4,r3,r2	R[4] ← PC; if (R[2] 0) PC← R[3]	9	4	3	2		plus

Format 6

6. add, sub, and, or

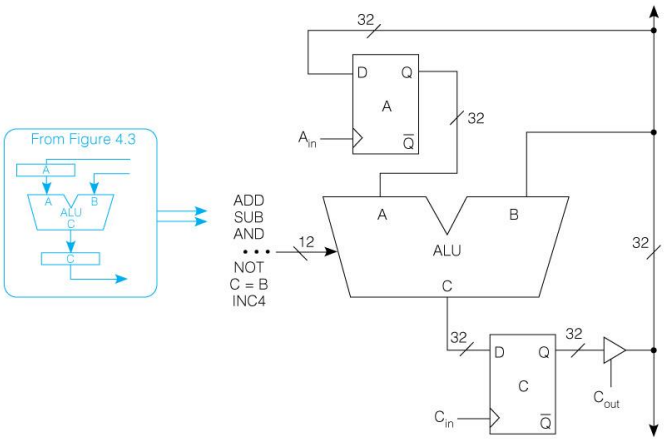
31 27	26 22	21 17	16 12	11 0
Op	ra	rb	rc	unused

- <op> ra,rb,rc R[ra]← R[rb] <op> R[rc]

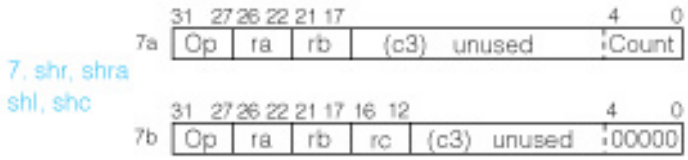
<op> = { add , sub, and, or } ;2's complement subtraction

add r0, r2, r4 (R[0] = R[2] + R[4])

Fig. 4.7 The ALU and Its Associated Registers



Format 7



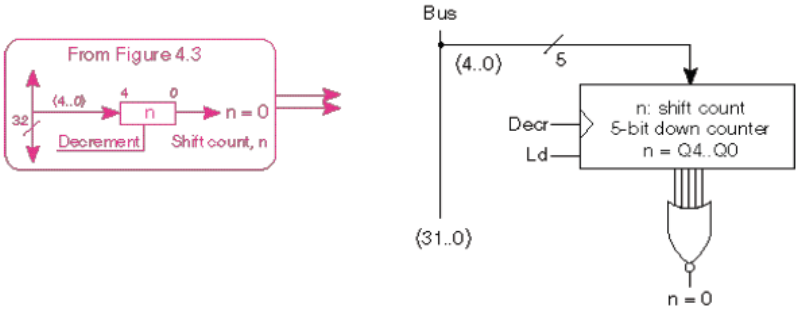
- תיאור סוגי ההזזות (shr,shra,shl,shc)
- 7(a) - בצע הזזה של R[rb] לפי המספר המידי המופיע בשדה <c3<4..0.
  המספר המוזז יושם ב R[ra]

shr ra,rb,count R[ra]← R[rb] Shifted right by count

- 7(b) - בצע הזזה של R[rb] לפי המספר המופיע ב R[rc].
  המספר המוזז יושם ב R[ra]

shr ra,rb,rc R[ra]← R[rb] Shifted right by count in R[rc]

Fig. 4.9 The Shift Counter



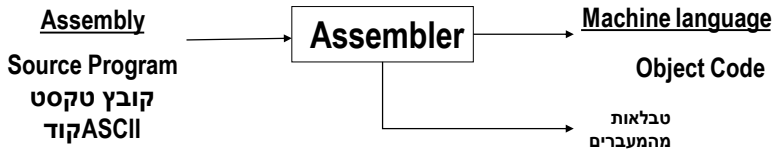
Pseudo-instructions

- הוראות לתוכנית האסמבלר – כיצד יש לתרגם את תוכנית האסמבלי לשפת מכונה.

- .org n - השורה הבאה של התוכנית תתחיל בכתובת n.
- .end - סוף תוכנית האסמבלי.
- .dc N - הקצה מילה בזיכרון עם הערך N.
- .dw m - הקצה מקום ל m מילים בזכרון (מערך).
- .equ M - תן ערך לשם סימבולי בתוכנית (דוגמא .equ 10 ten)

תוכנית האסמבלר.

- תוכנית האסמבלר : מקבלת שפה סימבולית (אסמבלי) ומתרגמת לייצוג שווה ערך בשפת המכונה.



- אסמבלר שני מעברים:
  - מעבר ראשון :
    - האסמבלר סורק את התוכנית ומשייך תוויות לכתובות
    - התוצר טבלה הממפה את התוויות לכתובות בזיכרון
  - מעבר שני
    - מבצעת השמה של הכתובות מהטבלה שנוצרה במעבר הראשון.
    - התוכנית מתורגמת לשפת מכונה.

Example of conditional branch

```
in C:   #define Cost 125
        if (X<0) then X = -X;
in SRC:
Cost    .equ  125          ;define symbolic constant
        .org  1000         ;next word will be loaded at address 1000_10
X:      .dw   1            ;reserve 1 word for variable X
        .org  5000         ;program will be loaded at location 5000_10
        lar   r31, Over    ;load address of "false" jump location
        ld    r1, X        ;load value of X into r1
        brpl  r31, r1      ;branch to Else if r1≥0
        neg   r1, r1       ;negate value
Over:   • • •              ;continue
```

דוגמא- תוכנית לביצוע xor בין שני משתנים והפיכתה לפונקציה

	.org 100	
A:	.dc 100	
B:	.dc 200	
Res :	.dw 1	
	.org 200	
xor:	ld r1, A	; R[1]←A
	not r2,r1	; R[2]←A’
	ld r3,b	; R[3]←B
	not r4,r3	; R[4]←B’
	and r5,r1,r4	; R[5]← A’B
	and r6,r2,r3	; R[6]← AB’
	or r7,r5,r6	; R[7]← AB’+A’B
	st r7,Res	Res←A xor B
	.end (br R11)	

	.org 300	
Main	lar r10,xor	; R[10]←Xor
	brl r11,r10	; R[11]←PC
		; PC←R[10]

דוגמא – תוכנית אסמבלי לחישוב $y=a^n$

```
%% Matlab Code y=a^n
%  a=5, n=3;

y=1;
for i=1:n
   x=0
   for j=1:y
      x=x+a;
   end
   y=x
end
```

```
%% SRC Assembly  Code y=a^n
; R1=y; R2=x; R3=a; R4=n

        la     r1,1        ; y=1;
        lar    r5, 0  ;
loop1:  lar    r2,0        ; x=0
        lar    r6,0
loop2:  add    r2,r2,r3    ; x=x+a
        addi   r1,r1,-1    ; y=y-1
        brnz   r6,r1;
        addi   r1,r2,0     ;y=x
        addi   r4,r4,-1    ; n=n-1
        bnnz   r5,r4
```