**VHDL.**

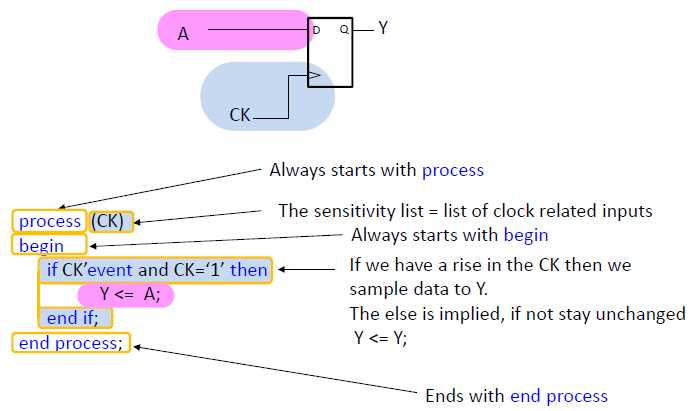
process (list of clock related inputs)

begin

all commands are here

if comes with end if; and elsif if needed

end process;



סיגנלים – signals.

* ביט בודד: signal NAME: std\_logic;.
* מספר ביטים: signal NAME: std\_logic\_vector (MSB down to 0).

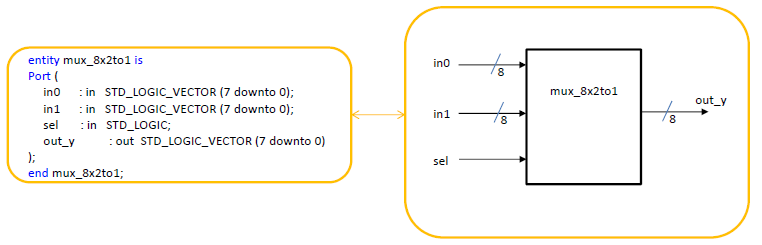
ישות – entity.

ישות מתארת את היציאות והכניסות של מכשיר או רכיב שרוצים להגדיר.

entity NAME is

Port (list of ins and outs);

end Name;



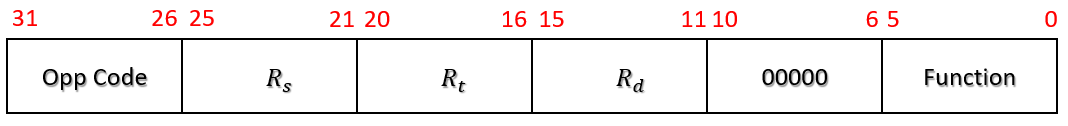
**MIPS' Assembler.**

רגיסטרים.

|  |  |  |
| --- | --- | --- |
| **שם** | **מספר** | **שימוש** |
| $zero | 0 | ערך קבוע 0 |
| $v0 - $v1 | 2-3 | ערכים עבור תוצאות והערכת ביטוי |
| $a0 - $a3 | 4-7 | ארגומנטים |
| $t3 - $t7 | 8-15 | זמניים |
| $s0 - $s7 | 16-23 | נשמרים |
| $t8 - $t9 | 24-25 | עוד זמניים |
| $gp | 28 | Global pointer |
| $sp | 29 | Stack pointer |
| $fp | 30 | Frame pointer |
| $ra | 31 | Return address |

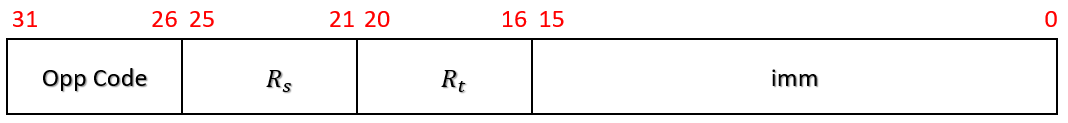
פקודות מסוג R-Type.

מכילה 32 ביטים, כאשר:

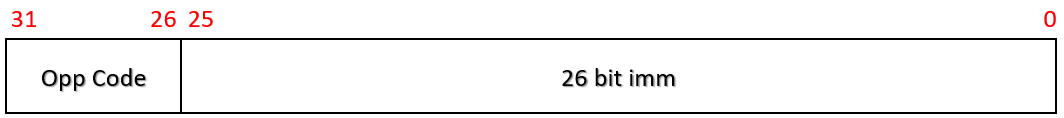


פקודות מסוג I-type.

מכילה 32 ביטים, כאשר:

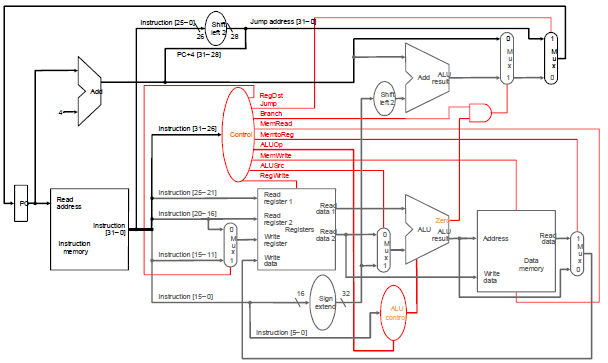


פקודות מסוג J-type.



שלבי ביצוע של המכונה.

1. Fetch – קריאת ההוראות מהזיכרון באמצעות ה-PC כפוינטר.
2. Decode – פיענוח ההוראות (החלטה מה לעשות בצעד הבא) וקריאת הרגיסטרים ההכרחיים (1 או 2).
3. Execute – חישוב התוצאה או כתובת הזיכרון באמצעות ה-ALU.
4. Memory – שימוש בתוצאת ה-ALU כדי לגשת לזיכרון אם צריך (קריאת נתונים, כתיבת נתונים).
5. Write back – אם צריך, כתיבת התוצאה לתוך הרגיסטר המתאים (עדכון רגיסטרים).



Codes of the Opcode fields – IR (31 downto 26):

|  |  |  |
| --- | --- | --- |
| **Command** | **Binary Opcode** | **Decimal Value** |
| sw | 101011 | 43 |
| lw | 100011 | 35 |
| lui | 001111 | 15 |
| ori | 001101 | 13 |
| addi | 001000 | 8 |
| beq | 000100 | 4 |
| bne | 000101 | 5 |
| j | 000010 | 2 |
| jal | 000011 | 3 |
| R-type | 000000 | 0 |

Functions field codes for R-type instructions – IR (5 down 0):

|  |  |  |
| --- | --- | --- |
| **Command** | **Binary Opcode** | **Decimal Value** |
| add | 100000 | 32 |
| sub | 100010 | 34 |
| and | 100100 | 36 |
| or | 100101 | 37 |
| xor | 100110 | 38 |
| slt | 101010 | 42 |
| jr | 001000 | 8 |

**תקציר פקודות.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **קטגוריה** | **פקודה** | **דוגמה** | **משמעות** | **הערות** |
| **אריתמטיות** | add | add $S1, $S2, $S3 | $S1 = $S2 + $S3 | המידע ברגיסטרים |
| sub | sub $S1, $S2, $S3 | $S1 = $S2 - $S3 | המידע ברגיסטרים |
| addi | addi $s1, $s2, 100 | $S1 = $S2 + 100 | הוספת קבוע |
| **העברת מידע** | lw | lw $s1, 100($s2) | $S1 = Memory[$S2 + 100] | מילה מהזיכרון לרגיסטר |
| sw | sw $s1, 100($s2) | Memory[$S2 + 100] = $S1 | מילה מרגיסטר לזיכרון |
| lb | lb $s1, 100($s2) | $S1 = Memory[$S2 + 100] | בייט מהזיכרון לרגיסטר |
| sb | sb $s1, 100($s2) | Memory[$S2 + 100] = $S1 | בייט מרגיסטר לזיכרון |
| lui | lui $s1, 100 | $S1 = 100 \* 216 | טעינת קבוע לתוך 16 ביטים עליונים |
| ori | ori $12, $0, 0xBEEF | $12 = 0x0000BEEF | טעינת קבוע לתוך 16 ביטים תחתונים |
| **קפיצה מותנית** | beq | beq $s1, $s2, 25 | if ($S1 == $S2) go to PC +4 + 100 | Branch on equal |
| bne | bne $s1, $s2, 25 | if ($S1 != $S2) go to PC +4 + 100 | Branch on not equal |
| slt | slt $s1, $s2, $s3 | if ($S2 < $S3) $S1 = 1;  else $S1 = 0; | Set on less than |
| slti | slti $s1, $s2, 100 | if ($S2 < 100) $S1 = 1;  else $S1 = 0; | Set on less than immediate |
| **קפיצה** | j | j 2500 | go to 10000 | קפיצה לכתובת |
| Jr | jr $ra | go to $ra | קפיצה לרגיסטר |
| jal | jal 2500 | $ra = PC + 4 ; go to 10000 | קפיצה לכתובת וחזרה לכתובת שאחרי הפקודה המקורית |

**דגשים חשובים.**

* GPR – general-purpose registers: כל 32 הרגיסטרים בעלי 32 ביטים של המעבד.
* בין כל שתי פקודות R-type צריך 2 nop כאשר משנים את אותו רגיסטר. אפשר לפתור את זה באמצעות data forwarding.
* Data forwarding – מעבירים מידע לשלב ה-EX של 1-3 צעדים אחורה. מתי צריך את זה? כאשר פונים לרגיסטר שכתבתנו אליו לפני 1-3 צעדים קודם.
* Branch forwarding – מעבירים מידע לשלב ה-ID.