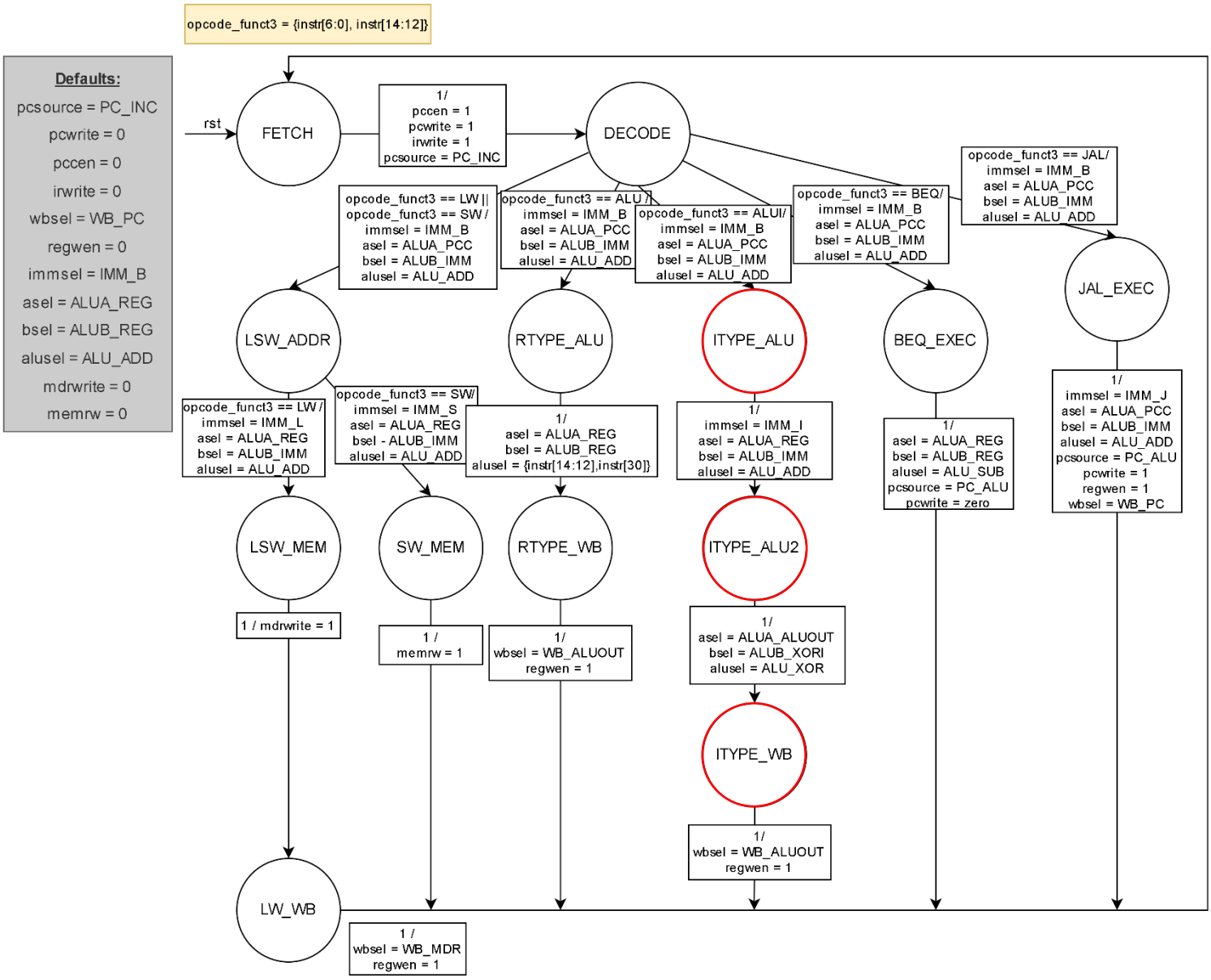
סימולציה 3

מוגש ע"י:

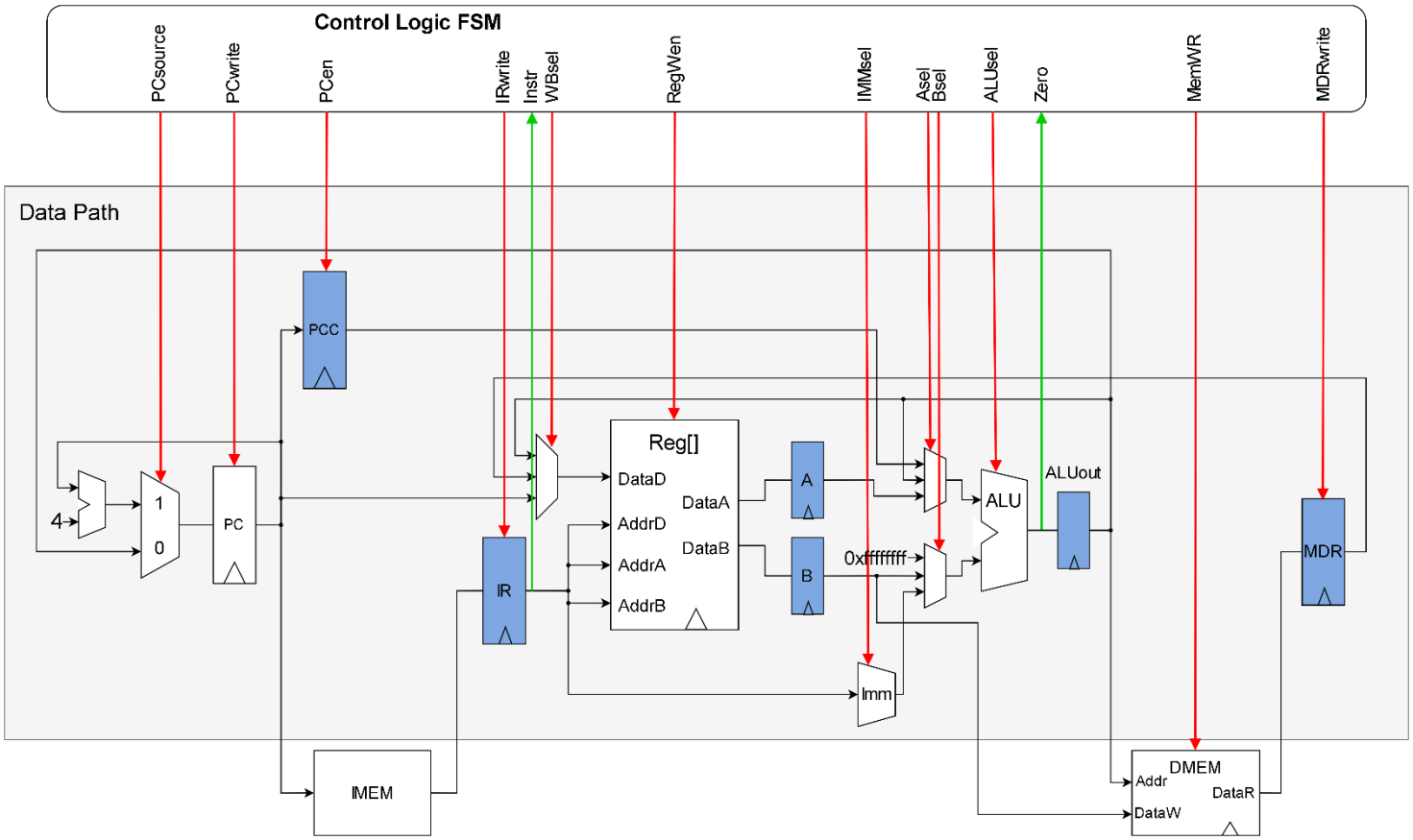
מייק אלכסנדרובסקי	315817155
דניאל טנסיינקו	318288925

2.4 –

בפקודת 'addi' ישנו שימוש ב-ALU פעמיים. נצטרך להוסיף למכונת המצבים שלושה מצבים חדשים שיהיו אחראיים על הפקודה.



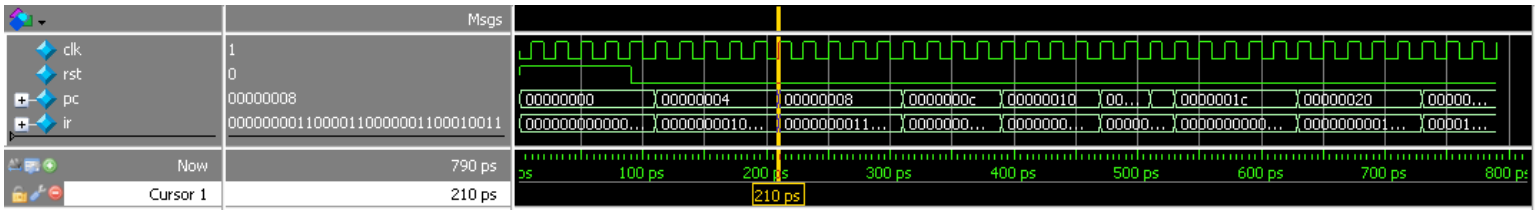
- ITYPE_ALU – מבצע את החיבור הראשוני בין rs1 ל-immediate ושומר את התוצאה ברגיסטר ALUout.
- ITYPE_ALU2 – מבצע את פעולת ה-xor בין התוצאה השמורה ב-ALUout לבין המספר 0xffffffff, ושומר את התוצאה ברגיסטר ALUout.
- ITYPE_WB – מבצע כתיבה חזרה של התוצאה הסופית השמורה ב-ALUout לרגיסטר rd.



השינויים שבוצעו במעבד:

- חיבור מוצא הרגיסטר ALUout ככניסה לבורר Asel.
- חיבור המספר 0xffffffff ככניסה לבורר Bsel.

2.3 –



ניתן לראות שב-91ps, rst יורד ל-0 והמעבד מתחיל לעבוד. בעליית השעון הבאה, הפקודה הראשונה נקראת (שלב ה-fetch) ו-pc מתעדכן ל-pc+4. הפקודה היא 'lw'.

ב-210ps, הפקודה הסתיימה והמעבד מקבל פקודה חדשה, גם כאן ps מתעדכן ועולה ב-4. ננתח את ir:

$$\underbrace{0000\ 0000\ 1100}_{\text{Imm (12)}}\ \underbrace{0011}_{\text{rs1 (6, x6=t1)}}\ \underbrace{0000}_{\text{func3}}\ \underbrace{0011}_{\text{rd}}\ \underbrace{0001\ 0011}_{\text{opcode}}$$

זוהי הפקודה 'addi t1, t1, 12'.

נחשב את התוצאה ונוודא שהיא תקינה:

t1 = 0xfffff453 (-2989)

t1 = t1 + 12 = 0xfffff45f (-2977)

t1 = t1 ^ 0xffffffff = 0x00000ba0 (-1120)

בהמשך מתקבלות הפקודות המתאימות ל-imem. לאחר ביצוע פקודת ה-'addi' תכולת הרגיסטר t1 נרשמת בזיכרון (dmem) בכתובת 16:

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/rv_sim/dmem
// format=hex addressradix=h dataradix=h version=1.0 wordsperline=1 noaddress
0000ff00
0000dead
fffff453
ffff0113
00000ba0
```

כפי שניתן לראות, הכתובת 16 מכילה את המספר שחושב.