29/3/2017

Αναφορά Εργαστηρίου 4

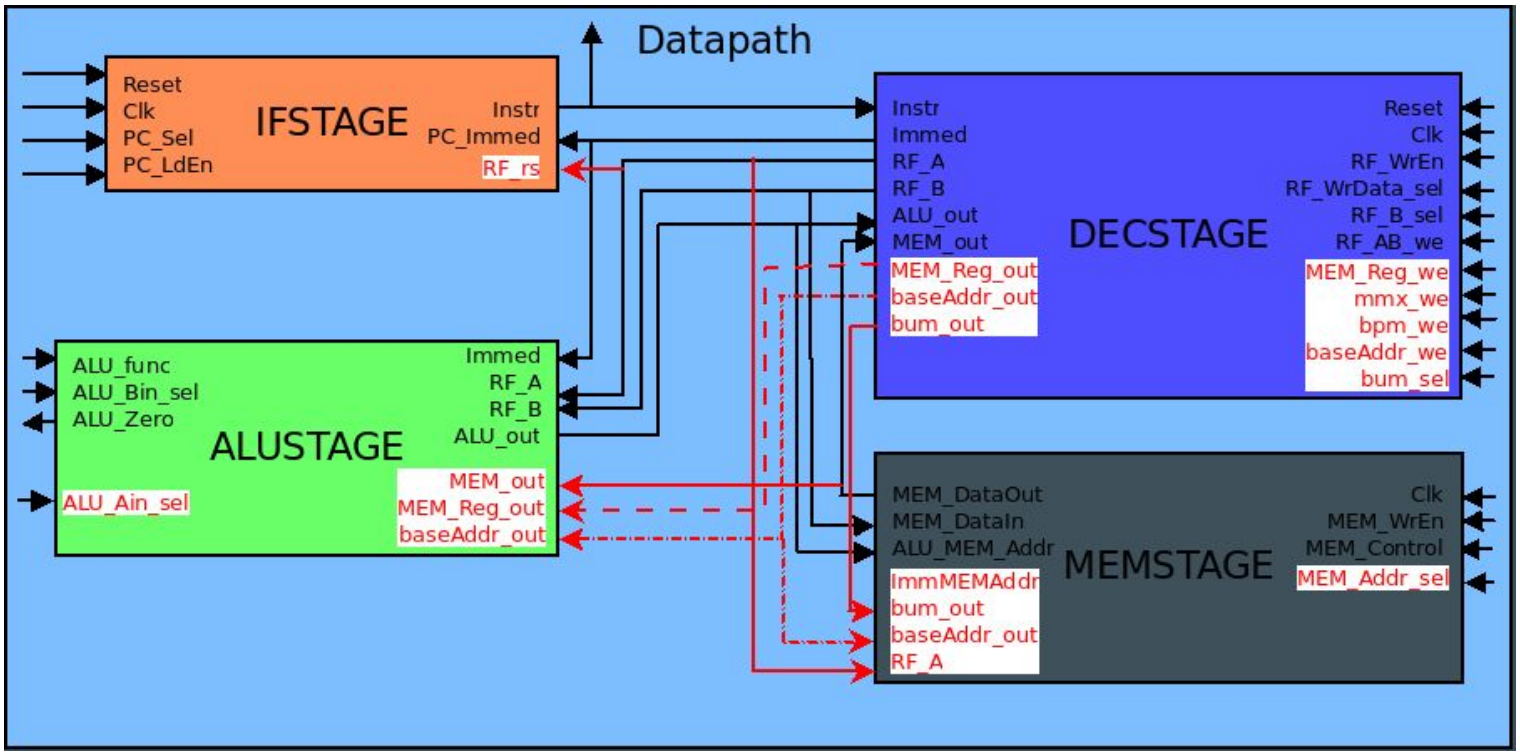
Οργάνωση Υπολογιστών

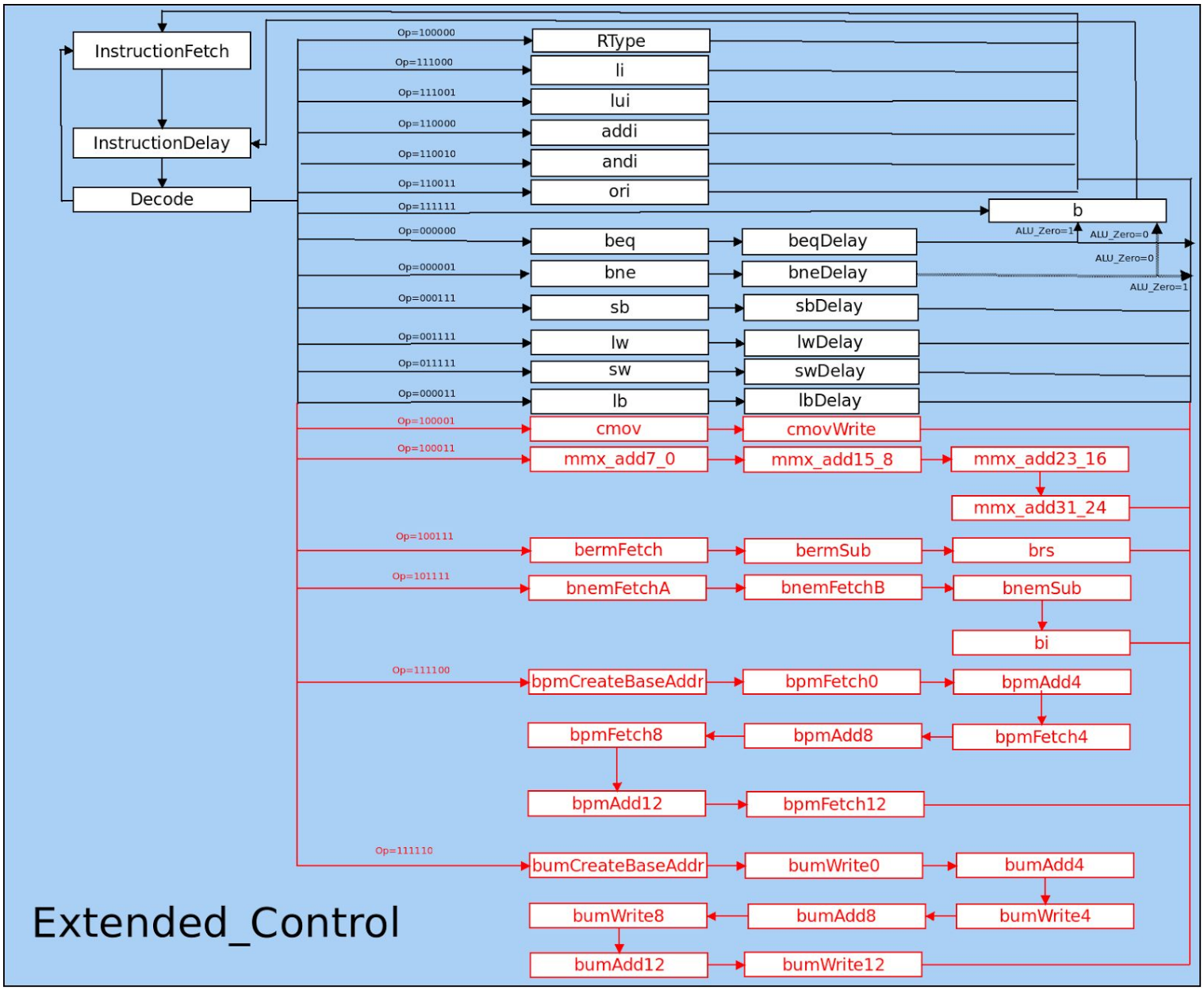
Ομάδα Εργασίας: Κριθαράκης Εμμανουήλ, Φωτάκης Τζανής

Κωδικός Ομάδας: LAB31231483

Προεργασία

Ως προεργασία του συγκεκριμένου εργαστηρίου είχε ζητηθεί όπως είναι φυσικό ο κώδικας των διαφόρων modules που περιγράφονται παρακάτω αλλά και σχεδιαγράμματα που παρουσιάζουν την συνδεσμολογία μεταξύ των διαφόρων ζητούμενων modules που κληθήκαμε να αναπτύξουμε. Παρακάτω παρατίθενται τα σχετικά σχεδιαγράμματα.





Περιγραφή της Άσκησης

Για το συγκεκριμένο εργαστήριο ζητήθηκε να προστεθούν κάποιες επιπλέον εντολές στο ήδη υπάρχων dataset του προηγούμενου εργαστηρίου. Συγκεκριμένα οι εντολές αυτές ήταν:

- ***Cmov*** :Σκοπός της εντολής αυτής ήταν να συγκρίνει αν περιέχει η τιμή του καταχωρητή RF[rt] οτιδήποτε διαφορετικό του μηδενός και εφόσον συνέβαινε κάτι τέτοιο τότε η τιμή του καταχωρητή RF[rs] αντιγράφονταν στον καταχωρητή RF[rd],αλλίως δεν εκτελούνταν τίποτα (no operation).Πάνω στην fsm η εντολή έπειτα απο το απαραίτητο decode της εισάγεται στο cmov state όπου και γίνεται ο έλενχος στην alu αν το περιεχόμενο του καταχωρητή RF[rt] είναι διαφορετικό απο το μηδεν(αφαίρεση με το μηδέν και έλενχος του alu_zero signal).Αν όντως ήταν διαφορετικό τότε στο cmovWrite stage πηγαίνουμε στο DECSTAGE κομμάτι του

datapath και εισάγουμε στον καταχωρητή RF[rd] το περιεχόμενο του RF[rs] αλλίως no-operation δηλαδή fetch της επόμενης εντολής του προγράμματος.(Προσοχή:για να έχουμε την δυνατότητα να συγκρίνουμε τον RF[rt] με την σταθερά 0 αυξήσαμε τον πολυπλέκτη εισόδου του A να δέχεται την σταθερή τιμή 0 με άμεση συνέπεια την αύξηση του control του πολυπλέκτη καθώς δεν υπήρχαν διαθέσιμες ελεύθερες θέσεις για input).

- ***add_MMX_byte*** *:*Σκοπός της εντολής αυτής ήταν να καταχωρήσει στον RF[rd] το αποτέλεσμα των προσθέσεων 4 8bits τμημάτων των καταχωρητών RF[rt] και RF[rs].Έπειτα απο τις προσθέσεις για να δημιουργηθεί μια 32bits ποσότητα τα αποτελέσματα περνούσαν απο μια λογική and και καταχωρούνταν στον RF[rd].Η προσέγγιση μας αποτελούνται απο 3 καταχωρητές που βρίσκονταν στο DECSTAGE κομμάτι όπου έπειτα απο την πρόσθεση μέσω της alu των 8bits τμημάτων των καταχωρητών RF[rs] και RF[rt] αποθηκεύονταν στον εκάστοτε καταχωρητή.Οι όποιες προσθέσεις και καταχωρήσεις έλαβαν μέρος στα mmx_add7_0, mmx_add15_8,mmx_add23_16,mmx_add31_24 stages.Στο τελευταίο stage όπου είχαμε διαθέσιμη την τελευταία πράξη της alu καθώς και τα αποθηκευμένα αποτελέσματα στους extra καταχωρητές στο DECSTAGE απλά αποθηκεύσαμε στον καταχωρητή RF[rd] το τελικό του αποτέλεσμα.(Προσοχή για την πρόσθεση στο ALUSTAGE οφείλαμε να έχουμε εισάγει στους πολυπλέκτες εισόδου της alu τις κατάλληλες 8bits ποσότητες).
- ***branch_equal_reg_mem*** *:*Σκοπός της εντολής αυτής ήταν να ελένχουμε την ισότητα του καταχωρητή RF[rt] αν είναι ίση με το περιεχόμενο της μνήμης του SignExtend(Imm<<2) και εφόσον είναι ίσα τότε θα προχωρούσα τον PC κατά PC+4+RF[rs] αλλίως κλασσικά ο program counter θα προχωρούσε στην επόμενη εντολή.Για τον έλενχο της ισότητας αρχικά πρέπει να κάνουμε fetch της ποσότητας στην μνήνη(bermFetch stage).Για να μπορέσουμε να κάνουμε signextend στην ποσότητα αυτή κάναμε update στον signExtender μας να κάνει sign extend και για αυτή την εντολή με το συγκεκριμένο opcode.Το αποτέλεσμα περνούσε με “καλώδιο” σε έναν απο τους πολυπλέκτες επιλογής τελεσταίου στην alu όπου και γίνονταν η αφαίρεση(bermSub stage) με τον RF[rt].Αν το alu_zero ήταν ενεργό τότε ο program counter προχωρούσε κατά +4+RF[rs] αλλίως προχωρούσε κλασσικά κατά +4.(brs stage) Για να μπορεί να προχωρήσει κατά +4+RF[rs] αυξήσαμε τις εισόδους στον πολυπλέκτη του IFSTAGEόπως και τα bits του control του.
- ***branch_not_equal_mem*** *:*Σκοπός της εντολής αυτής ήταν αντίστοιχος με τον παραπάνω με την διαφορά ότι ο έλενχος ισότητας ήταν μεταξύ των περιεχομένων της μνήμης μεταξύ RF[rs] και RF[rt] και αν δεν ήταν ίσα ο program counter προχωρούσε κατά +4+Immed αλλίως +4.Η fsm με παραπάνω διαφοροποιήται στο γεγονός οτι θέλω δύο stage για fetch στην μνήμη.(bnemFetchA stage),(bnemFetchB stage).Το πρώτο fetch αποθηκεύονταν σε καταχωρητή στο DECSTAGE για να μην χαθεί η τιμή του. Στους πολυπλέκτες επιλογής τελεσταίου στην alu όπου και γίνονταν η αφαίρεση(bnemSub stage) περνούσαν με “καλώδιο” τα αποτελέσματα των fetch.Αν το alu_Zero signal ήταν 0 τότε ο program counter προχωρούσε κατά +4+RF[rs] αλλίως προχωρούσε κλασσικά κατά +4.(bi stage)
- ***byte_pack_mem*** *:*Σκοπός της εντολής αυτής ήταν να καταχωρήσουμε στον RF[rd] τα 8bits lsb περιεχόμενα της μνήμης από 4 διαδοχικές διευθύνσεις στην μνήμη με την αρχική να υπολογίζεται ως το άθροισμα RF[rs] και SignExtend(Imm).Για την εντολή αυτή ενημερώσαμε τον signExtender να κάνει signExtend για το συγκεκριμένο opcode στο Immediate.Αρχικά στο (bpmCreateBaseAddr stage) δημιουργήσαμε το base_Addr με την πρόσθεση του RF[rs] και του SignExtend(Imm).Στην συνέχεια,ακολουθεί stage όπου κάνουμε fetch των 8 lsb bits στο base_Addr και τα αποθηκεύουμε σε 8 bits καταχωρητή (bpmFetch0).Ακολούθως προσθέτουμε στον base_Addr την τιμή +4 για την επόμενη διεύθυνση της μνήμης(bpmAdd4).Η παραπάνω διαδικασία επαναλαμβάνεται για τα stage (bpmFetch4,bpmAdd8,bpmFetch8,bpmAdd12,bpmFetch12) (Διευκρίνηση: με κάθε πρόσθεση προσθέτουμε το +4 καθώς αποθηκεύουμε σε καταχωρητή την τιμή του

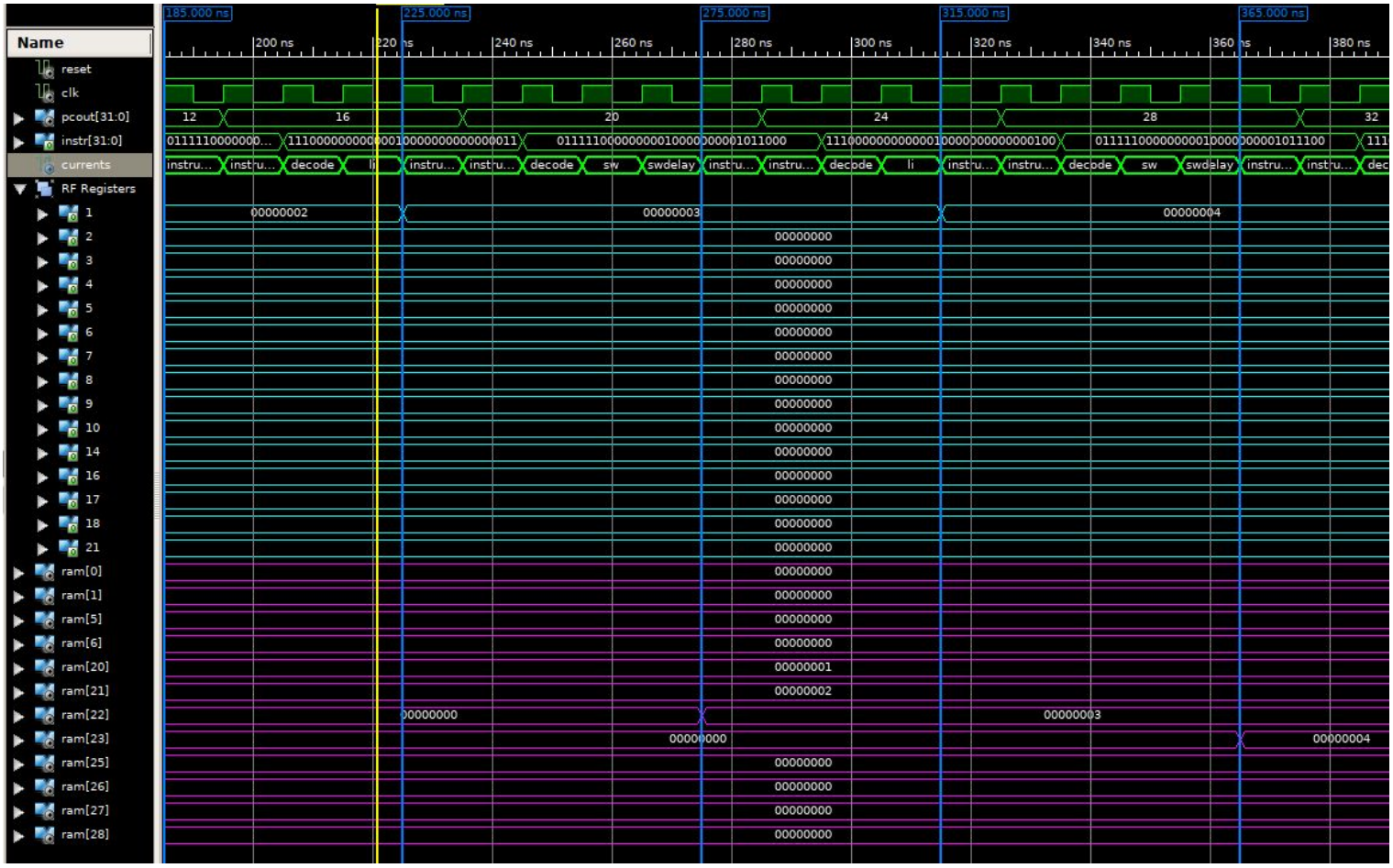
base_Addr έπειτα απο κάθε fetch.Για την πρόσθεση με την σταθερά +4 προσθέσαμε την σταθερά +4 ως δυνατή επιλογή τελεσταίου στους πολυπλέκτες επιλογής τελεσταίου στην alu στο ALUSTAGE).Τέλος στο stage bpmFetch12 έχοντας όλα τα ζητούμε 8bits αποτελέσματα τα καταχωρούμε στον RF[rd].

- ***byte_unpack_mem:***Σκοπός της εντολής αυτής ήταν να καταχωρούμε 4 8bit ποσότητες του RF[rd] έχοντας κάνει signExtend πρώτα, σε διαδοχικές θέσεις στην μνήμη με αρχική διεύθυνση την base_Addr η οποία προκύπτει όπως στην παραπάνω εντολή.Συνεπώς το πρώτο stage είναι γνωστό και πλέον ονομάζεται (bumCreateBaseAddr stage).Στην συνέχεια κάνουμε signExtend σε τμήματα του RF[rd] και τα γράφουμε στην μνήμη της διεύθυνσης base_Addr.(bumWrite0 stage).Ακολούθως προσθέτουμε την σταθερή τιμή +4 στο base_addr μέσω της alu και το αποτέλεσμα το αποθηκεύουμε σε καταχωρητή όπως και στην προηγούμενη εντολή(bumAdd4).Η παραπάνω διαδικασία επαναλαμβάνεται για τα stage (bumFetch4,bumAdd8,bumFetch8,bumAdd12,bumFetch12).Τέλος στο stage bumFetch12 αποθηκεύουμε τα τελευταία 8 bits του RF[rd](msb) στην μνήμη με διεύθυνση base_addr+12 (base_addr είναι η αρχική διεύθυνση)

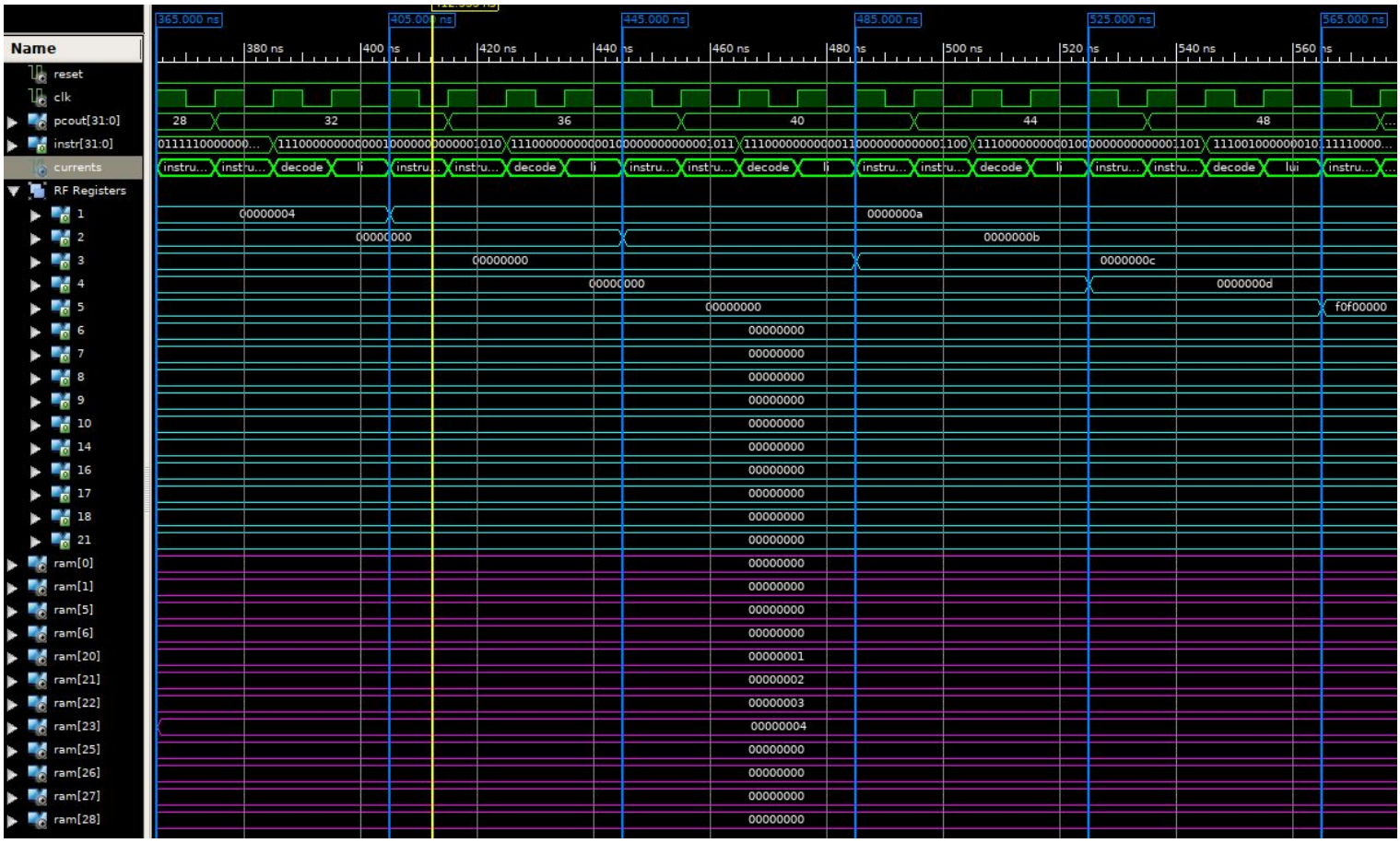
Κυμματομορφές

1. li $1, 1 -- $1 = 1
2. sw $1, 80($0) -- MEM[20] = 1
3. li $1, 2 -- $1 = 2

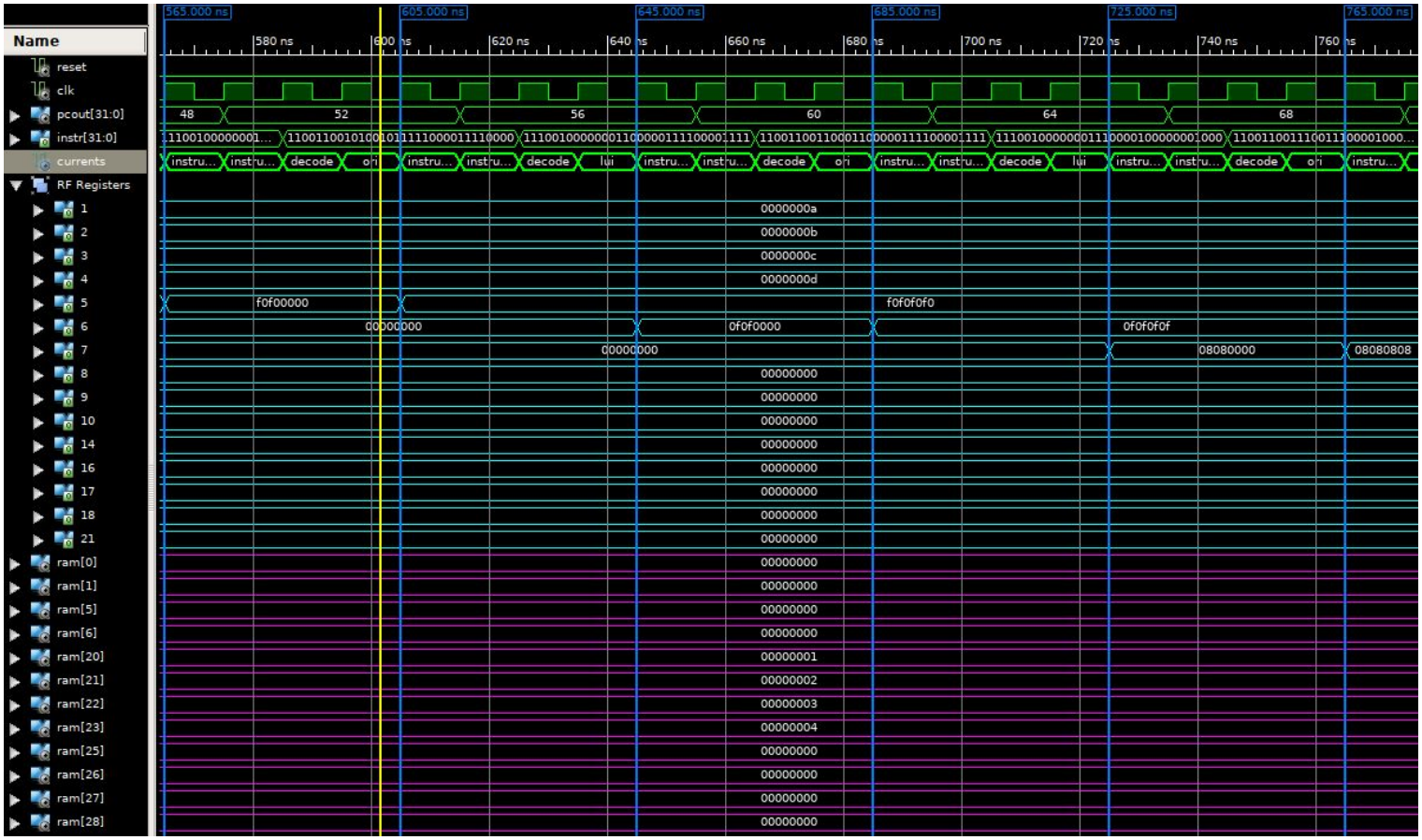
4. sw $1, 84($0) -- MEM[21] = 2



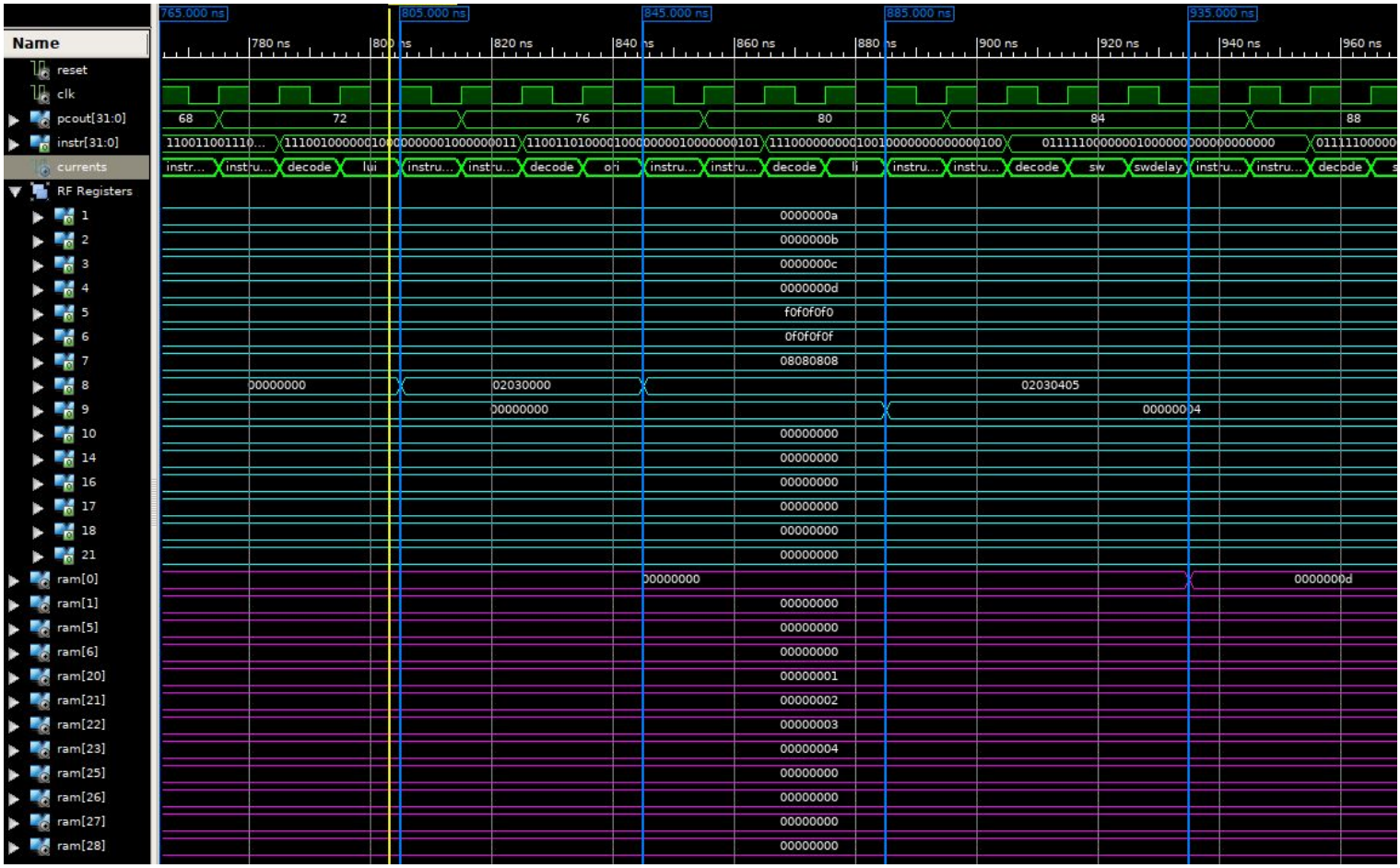
5. li $1, 3 -- $1 = 3
6. sw $1, 88($0) -- MEM[22] = 3
7. li $1, 4 -- $1 = 4
8. sw $1, 92($0) -- MEM[23] = 4



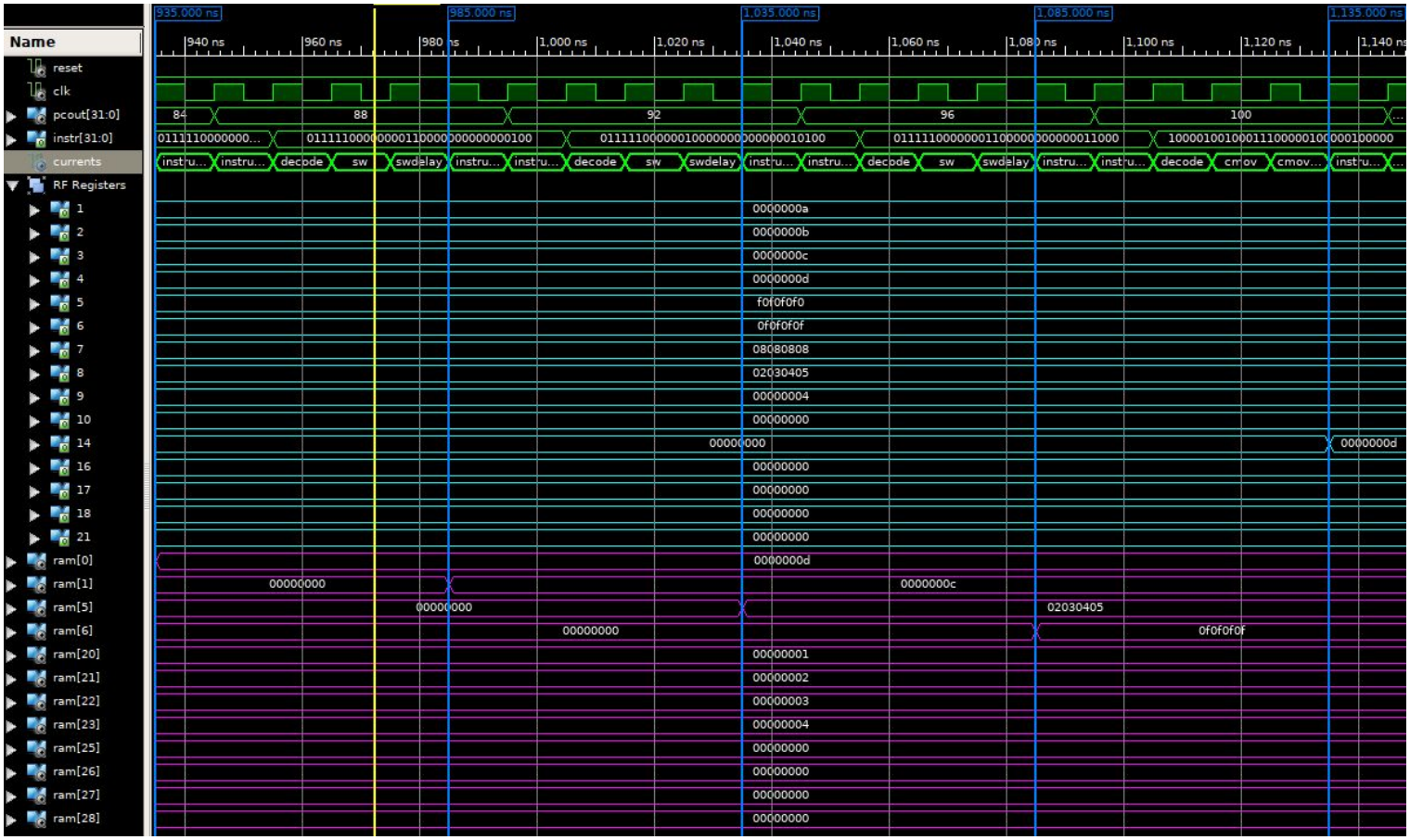
9. li $1, 10 -- $1 = 10
10. li $2, 11 -- $2 = 11
11. li $3, 12 -- $3 = 12
12. li $4, 13 -- $4 = 13
13. lui $5, x"F0F0" -- $5 = x"F0F00000"



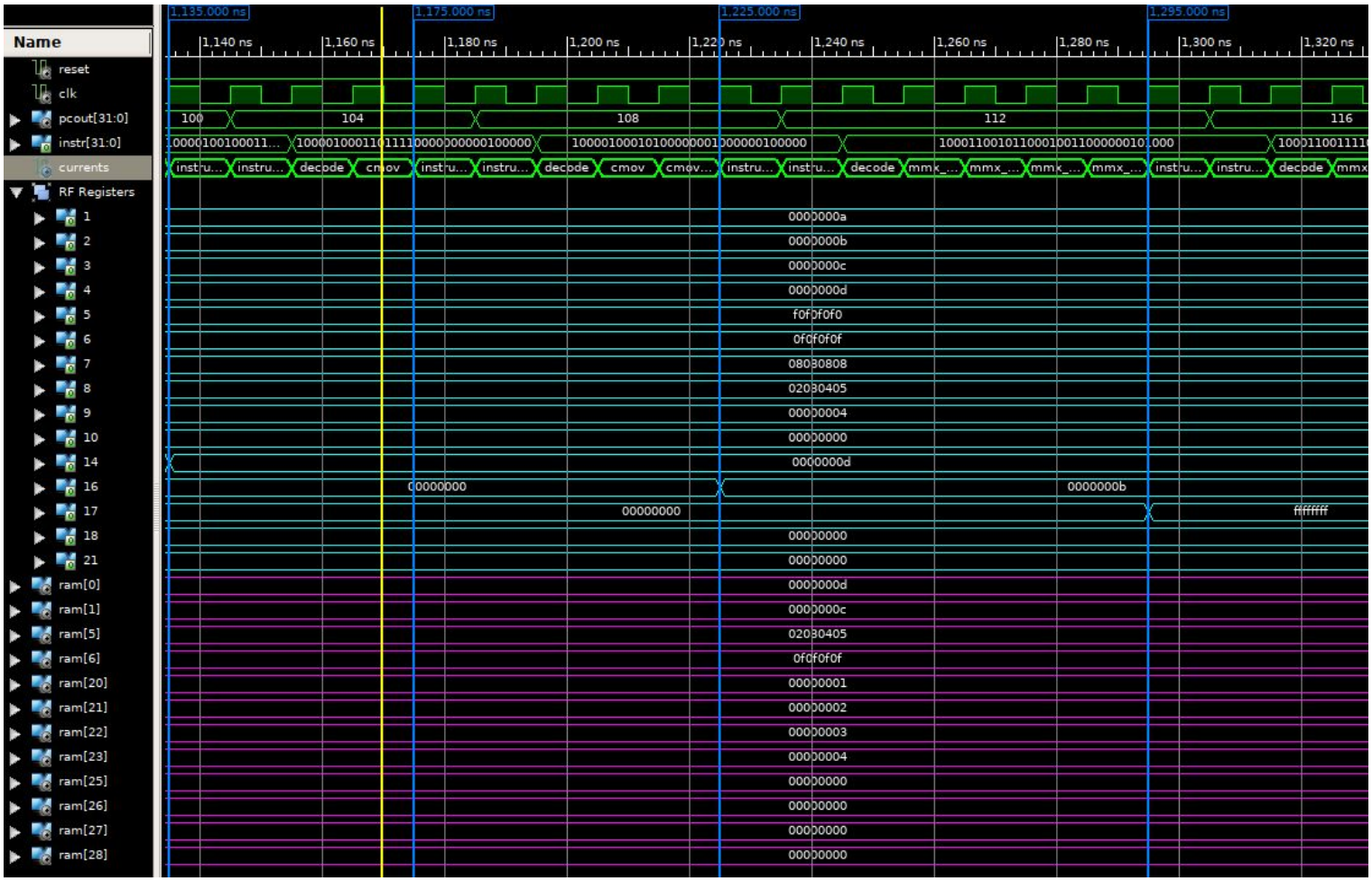
14. ori $5, $5, x"F0F0" -- $5 = x"F0F0F0F0"
15. lui $6, x"0F0F" -- $6 = x"0F0F0000"
16. ori $6, $6, x"0F0F" -- $6 = x"0F0F0F0F"
17. lui $7, x"0808" -- $7 = x"08080000"
18. ori $7, $7, x"0808" -- $7 = x"08080808"



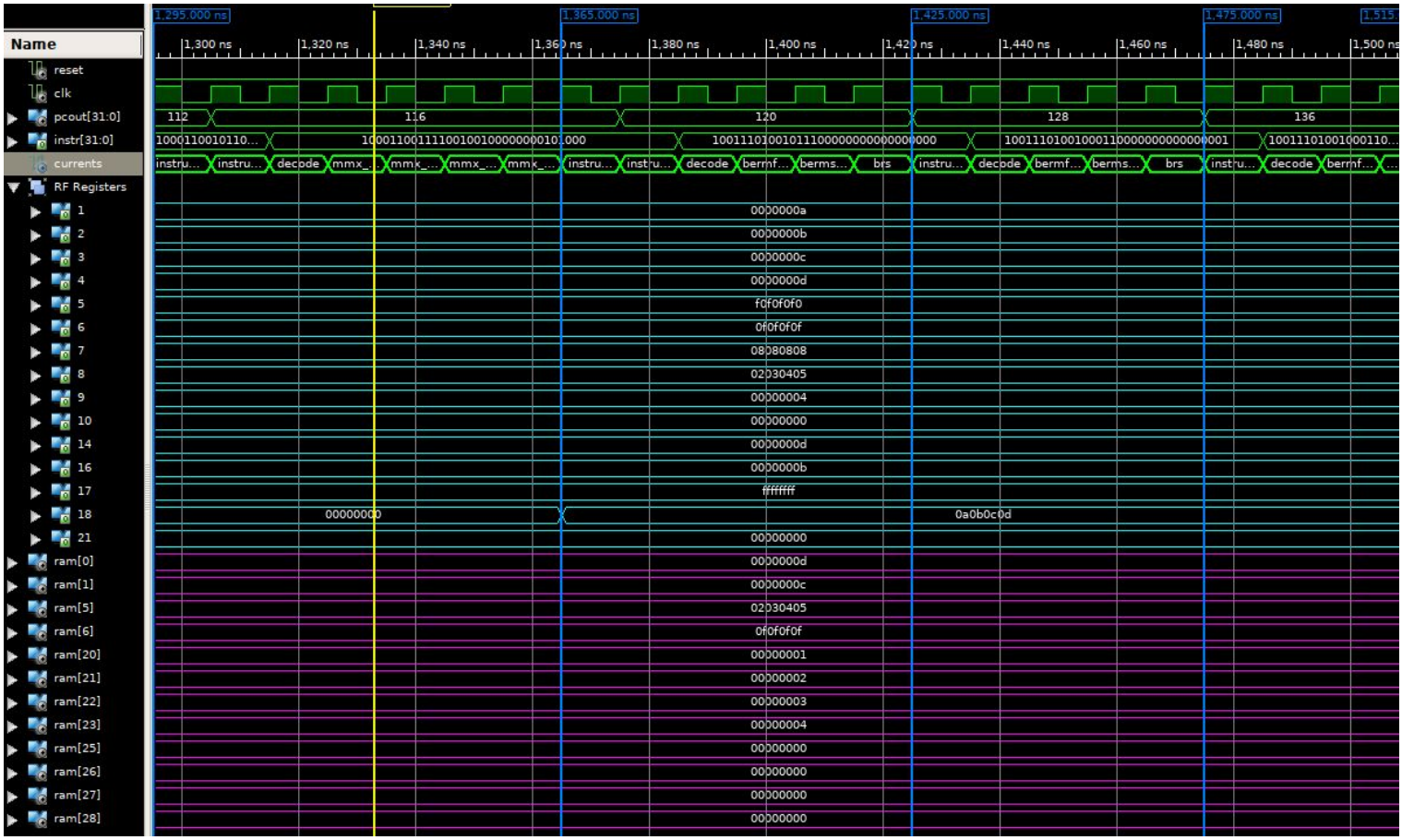
19. lui $8, x"0203" -- $8 = x"02030000"
20. ori $8, $8, x"0405" -- $8 = x"02030405"
21. li $9, 4 -- $9 = 4
22. sw $4, 0($0) -- MEM[0] = 13



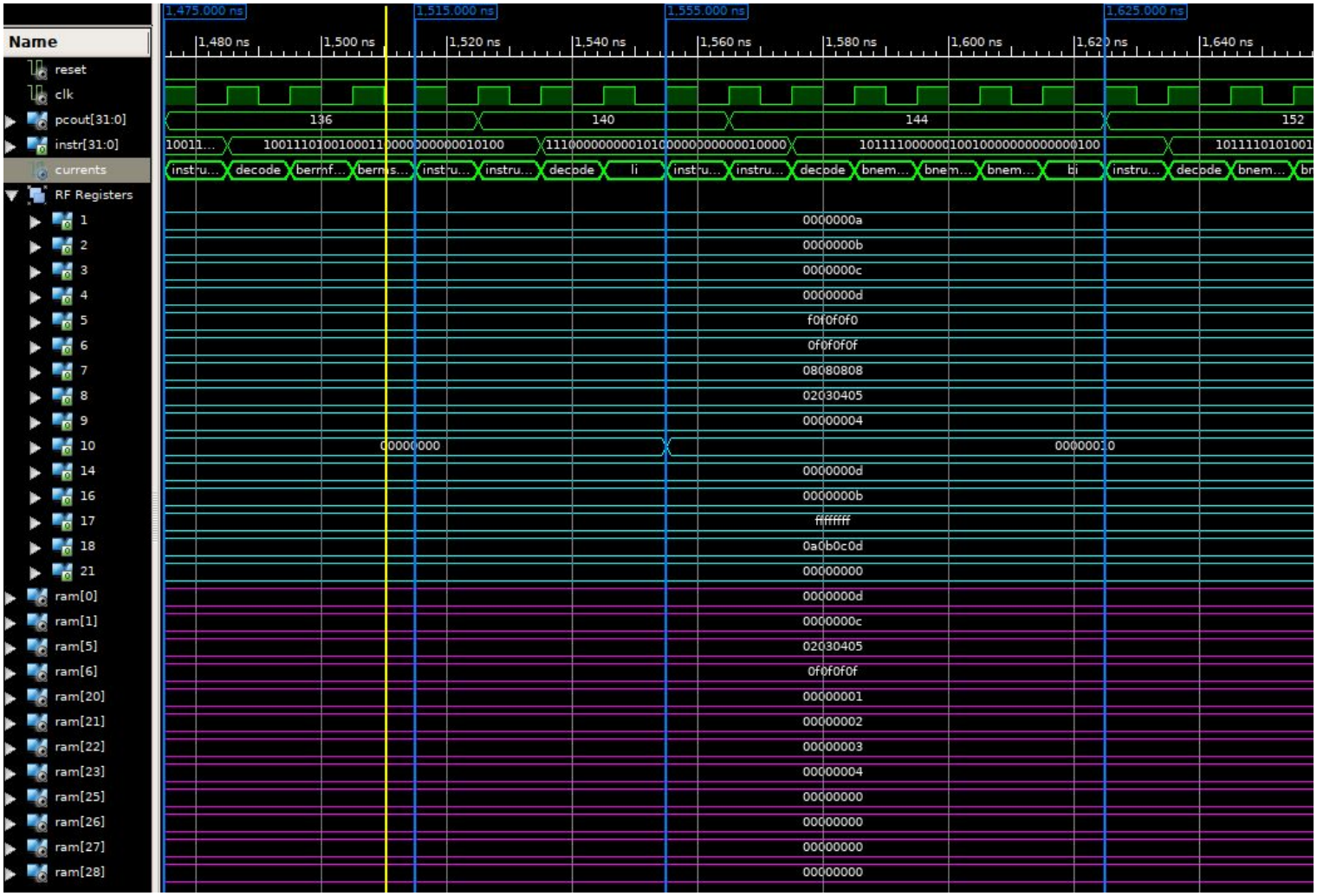
23. sw $3, 4($0) -- MEM[1] = 12
24. sw $8, 20($0) -- MEM[5] = x"02030405"
25. sw $6, 24($0) -- MEM[6] = x"0F0F0F0F"
26. cmov $4, $14, $1 -- $14 = 13



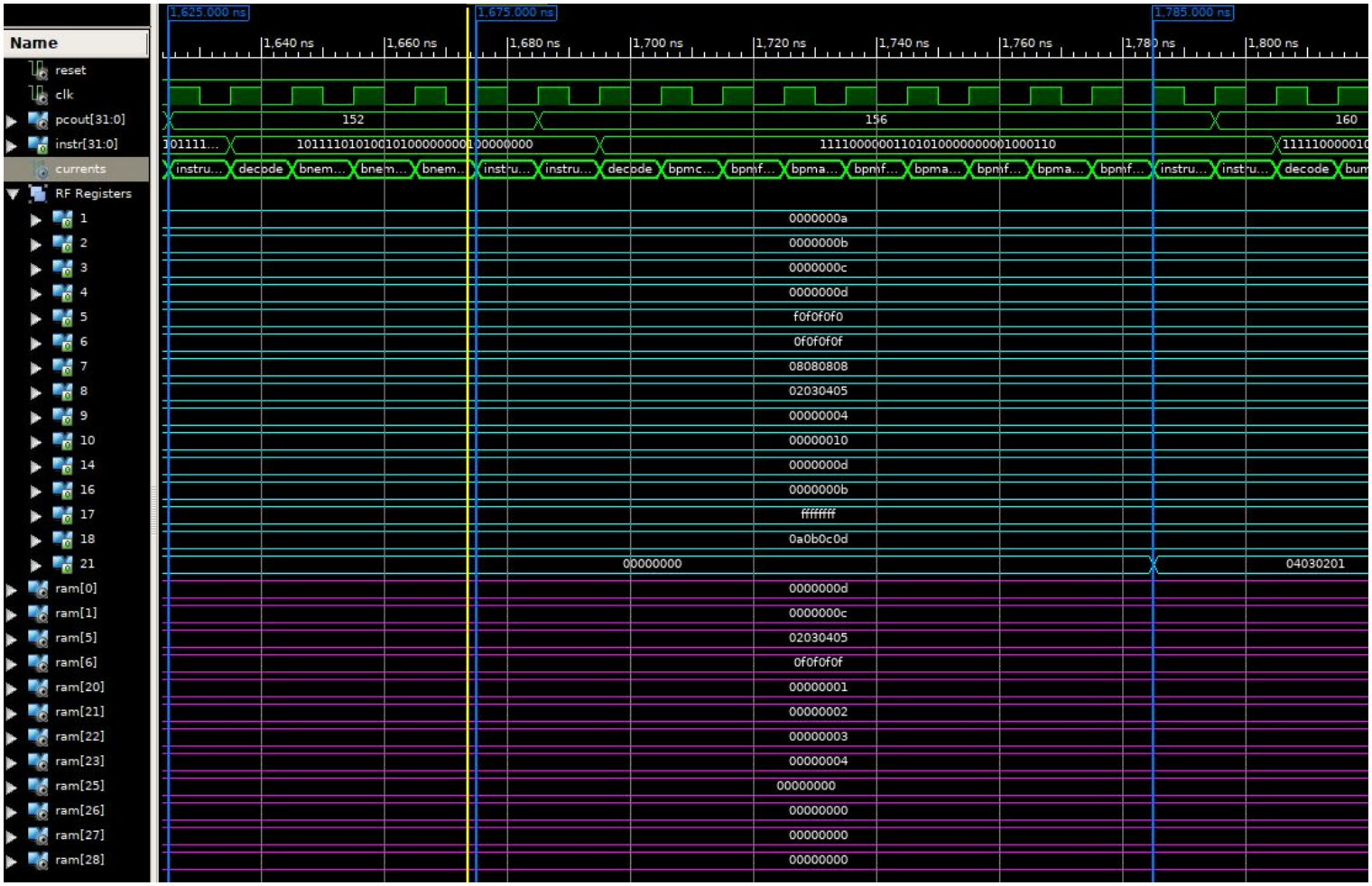
27. cmov $3, $15, $0 -- nop
28. cmov $2, $16, $2 -- $16 = 11
29. add_MMX_byte $5, $17, $6 -- $17 = x"FFFFFFFF"



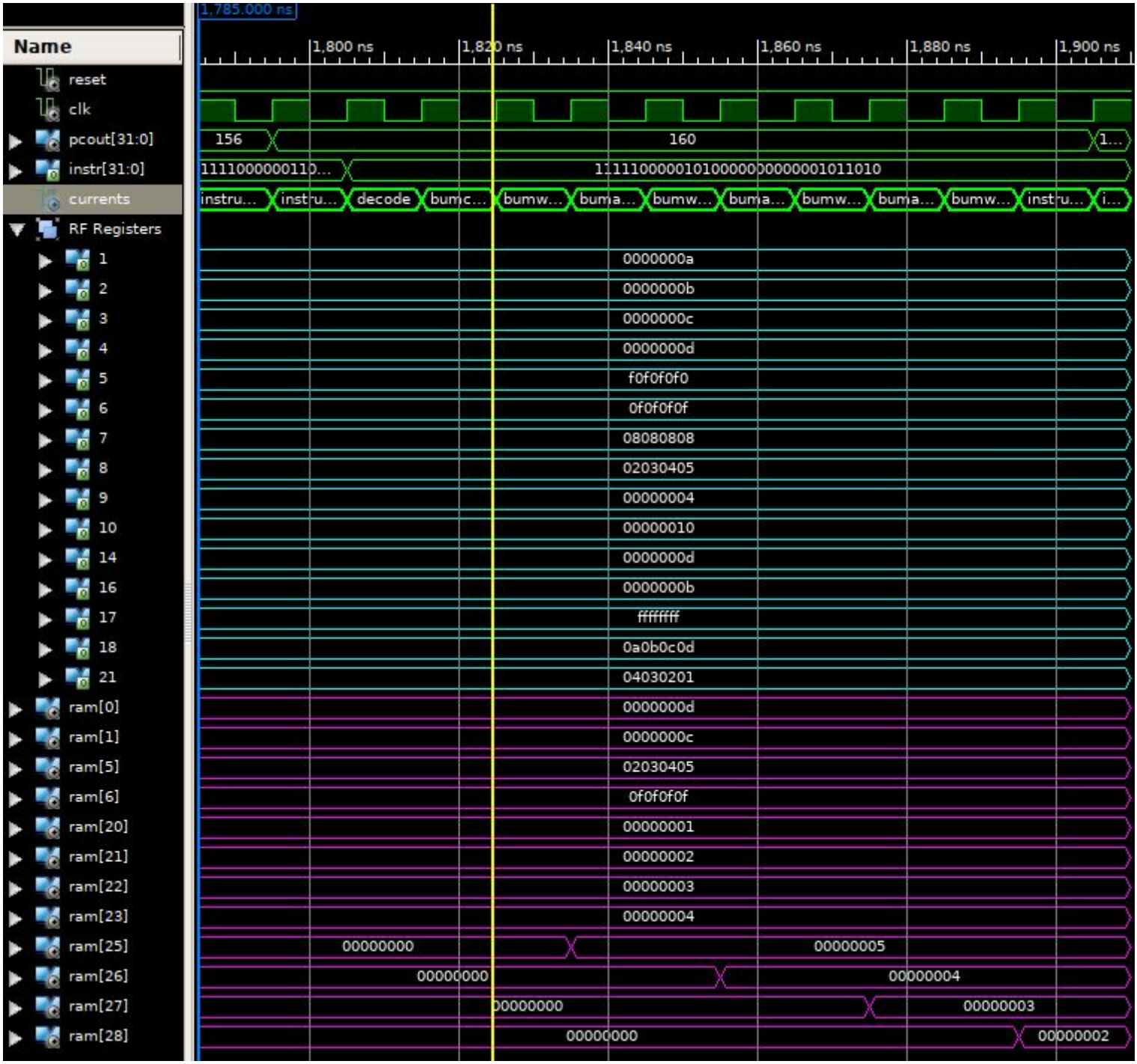
30. add_MMX_byte $7, $18, $8 -- $18 = x"0A0B0C0D"
31. branch_equal_reg_mem $9, $14, 0 -- PC = PC + 4 + 4 = 128
32. li $1, x"FFFF" -- NOP
33. branch_equal_reg_mem $9, $3, 1 -- PC = PC + 4 + 4 = 136
34. li $1, x"FFFF" -- NOP



35. branch_equal_reg_mem $9, $3, 20 -- PC = PC + 4 = 140
36. li $10, 16 -- $10 = 16
37. branch_not_equal_mem $0, $9, 4 -- PC = PC + 4 + 4 = 152
38. li $1, x"FFFF" -- NOP



39. branch_not_equal_mem $3, $3, 256 -- PC = PC + 4 = 156
40. byte_pack_mem $1, $21, 70 -- $21 = MEM[23](7:0)&MEM[22](7:0)&MEM[21](7:0)&MEM[20](7:0) = x"04030201"



41. byte_unpack_mem $1, $8, 90 -- MEM[25] = 5, MEM[26] = 4, MEM[27] = 3, MEM[28] = 2