



2025/12/11

## 實驗十三

姓名：林承羿

學號：01257027

班級：資工 3A

E-mail : IanLin6225@gmail.com

### ※注意

1. 繳交時一律轉 PDF 檔
2. 繳交期限為下周上課前
3. 一人繳交一份
4. 檔名請按照作業檔名格式進行填寫，未依照格式不予批改
5. 檔名範例：學號\_姓名\_HW13

### 一、組語撰寫練習—矩陣乘法

- 實驗說明：

1. 請透過當前課堂所學的所有組合語言撰寫一個程式計算兩個矩陣 A 和 B 的乘積，並將結果矩陣 C 儲存於記憶體位址 0x12 起的位置。矩陣 A 和 B 的元素皆為有號數 1 Byte，而矩陣 C 的元素為有號數 2 Bytes。程式需依據指定的矩陣儲存格式進行運算，如圖一所示。程式碼須加註解。

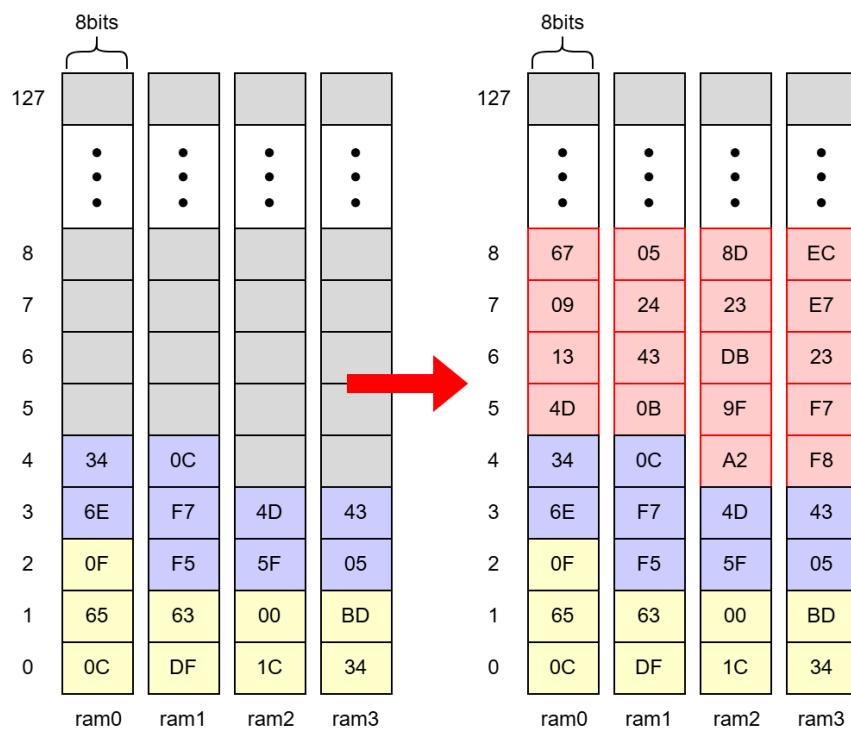
矩陣 A

$$\begin{bmatrix} 0C & DF & 1C \\ 34 & 65 & 63 \\ 00 & BD & 0F \end{bmatrix} \times \begin{bmatrix} F5 & 5F & 05 \\ 6E & F7 & 4D \\ 43 & 34 & 0C \end{bmatrix} = \begin{bmatrix} F8A2 & 0B4D & F79F \\ 4313 & 23DB & 2409 \\ E723 & 0567 & EC8D \end{bmatrix}$$

矩陣 B

矩陣 C

2. 所有暫存器皆可使用，但請將結果依序(0xF8A2→0x0B4D→...→0x0567→0xEC8D)載入到 x31 暫存器中。
3. 程式撰寫完成後透過 Tool Chain 將其轉換成 Program\_ROM.sv 檔，並使用 ModelSim 軟體觀察波形是否執行正確。
4. 請將所有用到的暫存器都加入至波形中以方便觀察。



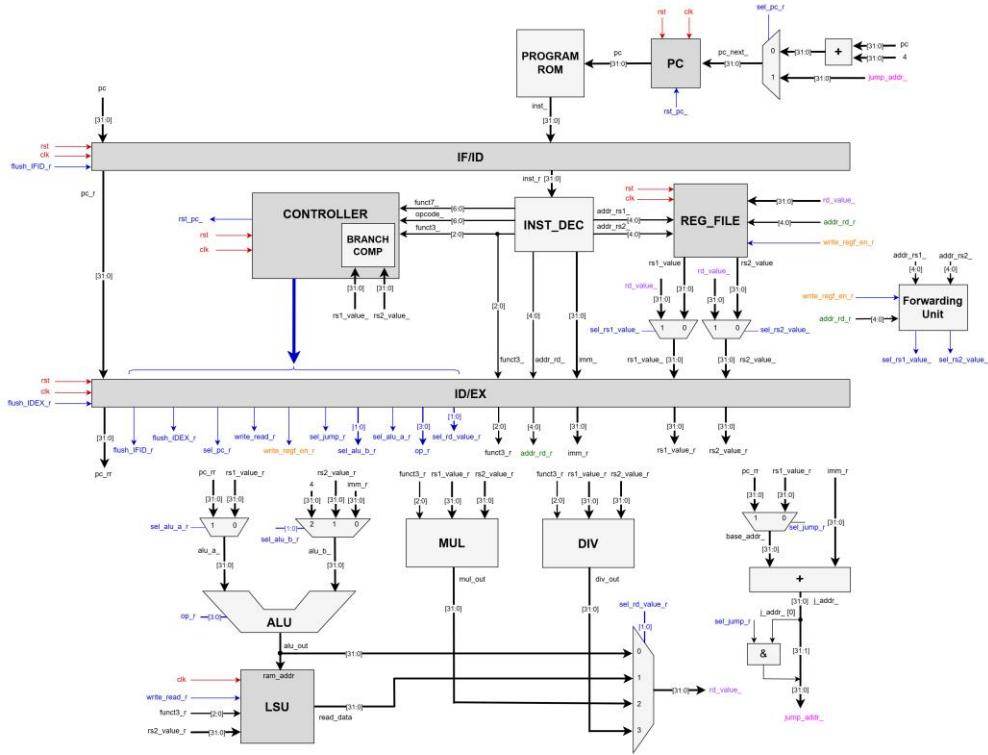
圖一

```
init:
    li x2, 0x341CDF0C
    sw x2, 0(x0)
    li x2, 0xBD006365
    sw x2, 4(x0)
    li x2, 0xF
    sb x2, 8(x0)
    li x3, 0x9
    li x2, 0x6E055FF5
    sw x2, 0(x3)
    li x2, 0x34434DF7
    sw x2, 4(x3)
    li x2, 0xC
    sb x2, 8(x3)

#-----開始撰寫您的組合語言-----
#-----您的組合語言-----
    li x3, 0x12
    li x4, 9
done:
    lh x31, 0(x3)
    addi x3, x3, 2
    addi x4, x4, -1
    bne x4, x0, done
```

圖二

## ● 系統硬體架構方塊圖（接線圖）：



## ● 系統架構程式碼、測試資料程式碼與程式碼說明

截圖請善用 win+shift+S

Controller.sv

```

controller.sv > controller
1 `include "mydefine.sv"
2 `timescale 1ns/100ps
3 module controller (
4     input logic [6:0] funct7_, opcode_,
5     input logic rst, clk,
6     input logic [2:0] funct3_,
7     input logic [31:0] rs1_value, rs2_value,
8     output logic flush_IFID_, flush_INDEX_, rst_pc_, sel_pc_, write_regf_en_, sel_jump_, sel_alu_a_, write_ram_,
9     output logic [1:0] sel_alu_b_, sel_rd_value_,
10    output logic [3:0] op_
11 );
12    logic BEQ_FLAG;
13    logic BNE_FLAG;
14    logic BLT_FLAG;
15    logic BGE_FLAG;
16    logic BLTU_FLAG;
17    logic BGEU_FLAG;
18
19    assign BEQ_FLAG = (rs1_value == rs2_value);
20    assign BNE_FLAG = (rs1_value != rs2_value);
21    assign BLT_FLAG = ($signed(rs1_value) < $signed(rs2_value));
22    assign BGE_FLAG = ($signed(rs1_value) >= $signed(rs2_value));
23    assign BLTU_FLAG = (rs1_value < rs2_value);
24    assign BGEU_FLAG = (rs1_value >= rs2_value);
25
26    typedef enum {s0, s1, s2} fsm_state;
27    fsm_state ps, ns;
28
29    always_ff @(posedge clk) begin
30        if (rst) begin
31            ps <= #1 s0;
32        end
33        else begin
34            ps <= #1 ns;
35        end
36    end
37
38    always_comb begin
39        rst_pc_ = 0;
40        sel_pc_ = 0;
41        flush_IFID_ = 0;
42        flush_INDEX_ = 0;
43        write_regf_en_ = 0;
44        sel_alu_b_ = 0;
45        ns = ps;
46        op_ = `ALUOP_ADD;
47        sel_alu_a_ = 0;
48        sel_jump_ = 0;
49        sel_rd_value_ = 0;
50        write_ram_ = 0;
51        unique case (ps)
52            s0: begin
53                flush_IFID_ = 1;
54                flush_INDEX_ = 1;
55                rst_pc_ = 1;
56                ns = s1;
57            end
58            s1: begin
59                flush_IFID_ = 1;
60                flush_INDEX_ = 1;
61                rst_pc_ = 1;
62                ns = s2;
63            end

```

```
64      s2: begin
65        case (opcode_ )
66          'Opcode_I: begin
67            unique case (funct3_ )
68              'F_ADDI: begin
69                op_ = 'ALUOP_ADD;
70                write_regf_en_ = 1;
71              end
72              'F_SLTI: begin
73                op_ = 'ALUOP_LT;
74                write_regf_en_ = 1;
75              end
76              'F_SLTIU: begin
77                op_ = 'ALUOP_LTU;
78                write_regf_en_ = 1;
79              end
80              'F_ANDI: begin
81                op_ = 'ALUOP_AND;
82                write_regf_en_ = 1;
83              end
84              'F_ORI: begin
85                op_ = 'ALUOP_OR;
86                write_regf_en_ = 1;
87              end
88              'F_XORI: begin
89                op_ = 'ALUOP_XOR;
90                write_regf_en_ = 1;
91              end
92              'F_SLLI: begin
93                op_ = 'ALUOP_SLL;
94                write_regf_en_ = 1;
95              end
96              'F_SRLI_SRAI: begin
97                unique case (funct7_ )
98                  'F7_SRLI: begin
99                    op_ = 'ALUOP_SRL;
100                   write_regf_en_ = 1;
101                 end
102                 'F7_SRAI: begin
103                   op_ = 'ALUOP_SRA;
104                   write_regf_en_ = 1;
105                 end
106               endcase
107             end
108           endcase
109         end
```

```

189
190     end
191     `Opcode_R_M: begin
192         unique case (funct3_)
193             `F_AND: begin
194                 if (funct7_ == `F7_OPCODE_R) begin
195                     op_ = `ALUOP_AND;
196                     write_regf_en_ = 1;
197                     sel_alu_b_ = 1;
198                 end
199                 // `F_REMU
200                 else if (funct7_ == `F7_R_M) begin
201                     write_regf_en_ = 1;
202                     sel_rd_value_ = 3;
203                 end
204             end
205             `F_OR: begin
206                 if (funct7_ == `F7_OPCODE_R) begin
207                     op_ = `ALUOP_OR;
208                     write_regf_en_ = 1;
209                     sel_alu_b_ = 1;
210                 end
211                 // `F_REM
212                 else if (funct7_ == `F7_R_M) begin
213                     write_regf_en_ = 1;
214                     sel_rd_value_ = 3;
215                 end
216             end
217             `F_XOR: begin
218                 if (funct7_ == `F7_OPCODE_R) begin
219                     op_ = `ALUOP_XOR;
220                     write_regf_en_ = 1;
221                     sel_alu_b_ = 1;
222                 end
223                 // `F_DIV
224                 else if (funct7_ == `F7_R_M) begin
225                     write_regf_en_ = 1;
226                     sel_rd_value_ = 3;
227                 end
228             end
229         endcase
230     end
231     `F_ADD_SUB: begin
232         unique case (funct7_)
233             `F7_ADD: begin
234                 op_ = `ALUOP_ADD;
235                 write_regf_en_ = 1;
236                 sel_alu_b_ = 1;
237             end
238             `F7_SUB: begin
239                 op_ = `ALUOP_SUB;
240                 write_regf_en_ = 1;
241                 sel_alu_b_ = 1;
242             end
243             // `MUL
244             `F7_R_M: begin
245                 write_regf_en_ = 1;
246                 sel_rd_value_ = 2;
247             end
248         endcase
249     end
250     `F_SLT: begin
251         if (funct7_ == `F7_OPCODE_R) begin
252             op_ = `ALUOP_LT;
253             write_regf_en_ = 1;
254             sel_alu_b_ = 1;
255         end
256         // `F_MULHSU
257         else if (funct7_ == `F7_R_M) begin
258             write_regf_en_ = 1;
259             sel_rd_value_ = 2;
260         end
261     end
262 
```

```

179      `F_SLTU: begin
180        if (funct7_ == `F7_OPCODE_R) begin
181          op_ = `ALUOP_LTU;
182          write_regf_en_ = 1;
183          sel_alu_b_ = 1;
184        end
185        // F_MULHU
186        else if (funct7_ == `F7_R_M) begin
187          write_regf_en_ = 1;
188          sel_rd_value_ = 2;
189        end
190      end
191      `F_SLL: begin
192        if (funct7_ == `F7_OPCODE_R) begin
193          op_ = `ALUOP_SLL;
194          write_regf_en_ = 1;
195          sel_alu_b_ = 1;
196        end
197        // F_MULH
198        else if (funct7_ == `F7_R_M) begin
199          write_regf_en_ = 1;
200          sel_rd_value_ = 2;
201        end
202      end
203      `F_SRL_SRA: begin
204        unique case (funct7_)
205          `F7_OPCODE_R: begin
206            op_ = `ALUOP_SRL; // SRL
207            write_regf_en_ = 1;
208            sel_alu_b_ = 1;
209          end
210          `F7_SRA: begin
211            op_ = `ALUOP_SRA; // SRA (if different funct7)
212            write_regf_en_ = 1;
213            sel_alu_b_ = 1;
214          end
215          // F_DIVU
216          `F7_R_M: begin
217            write_regf_en_ = 1;
218            sel_rd_value_ = 3;
219          end
220        endcase
221      end
222    endcase
223  end
224  `Opcode_B: begin
225    sel_jump_ = 1;
226    unique case (funct3_)
227      `F_BEQ: begin
228        if (BEQ_FLAG) begin
229          sel_pc_ = 1;
230          flush_INDEX_ = 1;
231          flush_IFID_ = 1;
232        end
233      end
234      `F_BNE: begin
235        if (BNE_FLAG) begin
236          sel_pc_ = 1;
237          flush_INDEX_ = 1;
238          flush_IFID_ = 1;
239        end
240      end
241      `F_BLT: begin
242        if (BLT_FLAG) begin
243          sel_pc_ = 1;
244          flush_IFID_ = 1;
245          flush_INDEX_ = 1;
246        end
247      end
248      `F_BGE: begin
249        if (BGE_FLAG) begin
250          sel_pc_ = 1;
251          flush_INDEX_ = 1;
252          flush_IFID_ = 1;
253        end
254      end

```

```

254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1196
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1267
1268
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1317
1318
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1327
1328
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1367
1368
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1417
1418
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1767
1768
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1777
1778
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1867
1868
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1917
1918
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1927
1928
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1937
1938
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1947
1948
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1967
1968
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1977
1978
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2017
2018
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2027
2028
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2037
2038
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2047
2048
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2067
2068
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2077
2078
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2117
2118
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2127
2128
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2137
2138
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2147
2148
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2167
2168
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2177
2178
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2187
2188
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2217
2218
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2227
2228
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2237
2238
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2247
2248
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2267
2268
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2277
2278
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2287
2288
2288
2289
2289

```

```
331      Uopcode_S; begin
332          unique case (funct3_)
333              'F_SB: begin
334                  op_ = `ALUOP_ADD;
335                  write_ram_ = 1;
336              end
337              'F_SH: begin
338                  op_ = `ALUOP_ADD;
339                  write_ram_ = 1;
340              end
341              'F_SW: begin
342                  op_ = `ALUOP_ADD;
343                  write_ram_ = 1;
344              end
345          endcase
346      end
347  endcase
348 end
349 endcase
350 end
351 endmodule
```

## div.sv

```
1  `include "mydefine.sv"
2  module DIV (
3      input logic [2:0] funct3,
4      input logic [31:0] rs1_value,
5      input logic [31:0] rs2_value,
6
7      output logic [31:0] div_out
8  );
9
10    always_comb begin
11        unique case (funct3)
12            'F_DIV: div_out = $signed(rs1_value) / $signed(rs2_value);
13            'F_DIVU: div_out = rs1_value / rs2_value;
14            'F_REM: div_out = $signed(rs1_value) % $signed(rs2_value);
15            'F_REMU: div_out = rs1_value % rs2_value;
16        endcase
17    end
18
19 endmodule
```

## Inst\_dec.sv

```

INST_DEC.sv>-
1 `include "mydefine.sv"
2 module INST_DEC (
3     input logic [31:0] inst_r,
4     output logic [6:0] funct7_, opcode_,
5     output logic [4:0] addr_rs1_, addr_rs2_, addr_rd_,
6     output logic [2:0] funct3_,
7     output logic [31:0] imm_
8 );
9     assign opcode_ = inst_r[6:0];
10    assign funct3_ = inst_r[14:12];
11    assign addr_rd_ = inst_r[11:7];
12    assign addr_rs1_ = inst_r[19:15];
13    assign addr_rs2_ = inst_r[24:20];
14    assign funct7_ = inst_r[31:25];
15
16    logic [31:0] IMM_I;
17    logic [31:0] IMM_B;
18    logic [31:0] IMM_JAL;
19    logic [31:0] IMM_LUI_AUIPC;
20    logic [31:0] IMM_S;
21    assign IMM_I = {({20{inst_r[31]}}, inst_r[31:20])};
22    assign IMM_B = {({20{inst_r[31]}}, inst_r[7], inst_r[30:25], inst_r[11:8], 1'b0)};
23    assign IMM_JAL = {({12{inst_r[31]}}, inst_r[19:12], inst_r[20], inst_r[30:21], 1'b0)};
24    assign IMM_LUI_AUIPC = {inst_r[31:12], 12'b0};
25    assign IMM_S = {({20{inst_r[31]}}, inst_r[31:25], inst_r[11:7])};
26
27    always_comb begin
28        unique case (opcode_)
29            'Opcode_I: imm_ = IMM_I;
30            'Opcode_B: imm_ = IMM_B;
31            'Opcode_JAL: imm_ = IMM_JAL;
32            'Opcode_JALR: imm_ = IMM_I;
33            'Opcode_LUI: imm_ = IMM_LUI_AUIPC;
34            'Opcode_AUIPC: imm_ = IMM_LUI_AUIPC;
35            'Opcode_L: imm_ = IMM_I;
36            'Opcode_S: imm_ = IMM_S;
37        endcase
38    end
39
40 endmodule

```

## Mul.sv

```

MUL.sv>-
1 `include "mydefine.sv"
2 module MUL (
3     input logic [2:0] funct3,
4     input logic [31:0] rs1_value,
5     input logic [31:0] rs2_value,
6
7     output logic [31:0] mul_out
8 );
9     logic [63:0] product;
10
11    always_comb begin
12        unique case (funct3)
13            'F_MUL: product = $signed(rs1_value) * $signed(rs2_value);
14            'F_MULH: product = $signed(rs1_value) * $signed(rs2_value);
15            'F_MULHSU: product = $signed($signed(rs1_value) * rs2_value);
16            'F_MULHU: product = rs1_value * rs2_value;
17        endcase
18    end
19
20    assign mul_out = (funct3 == 'F_MUL) ? product[31:0] : product[63:32];
21
22 endmodule

```

## Myalu.sv

```
⑤ myalu.sv > ...
1 `include "mydefine.sv"
2 module myalu (
3     input logic [3:0] op,
4     input logic [31:0] alu_a,
5     input logic [31:0] alu_b,
6     output logic [31:0] alu_out
7 );
8     always_comb begin
9         unique case (op)
10             `ALUOP_ADD : alu_out = alu_a + alu_b;
11             `ALUOP_SUB : alu_out = $signed(alu_a) - $signed(alu_b);
12             `ALUOP_AND : alu_out = alu_a & alu_b;
13             `ALUOP_OR : alu_out = alu_a | alu_b;
14             `ALUOP_XOR : alu_out = alu_a ^ alu_b;
15             `ALUOP_A : alu_out = alu_a;
16             `ALUOP_A_ADD_4 : alu_out = alu_a + 4;
17             `ALUOP_LTU : alu_out = alu_a < alu_b;
18             `ALUOP_LT : alu_out = $signed(alu_a) < $signed(alu_b);
19             `ALUOP_SLL : alu_out = alu_a << alu_b[4:0];
20             `ALUOP_SRL : alu_out = alu_a >> alu_b[4:0];
21             `ALUOP_SRA : alu_out = $signed(alu_a) >>> alu_b[4:0];
22             `ALUOP_B : alu_out = alu_b;
23             default : alu_out = alu_a;
24         endcase
25     end
26 endmodule
```

## Mycpu.sv

```
mycpu.sv > ...
1 `include "mydefine.sv"
2 module mycpu(
3     input logic clk, rst,
4     output logic [31:0] regs_31
5 );
6 // program-counter
7 logic rst_pc_, sel_pc_r;
8 logic [31:0] pc, pc_next, pc_r, pc_rr, jump_addr_;
9 always_comb begin
10     case (sel_pc_r)
11         1'd0: begin
12             pc_next = pc + 4;
13         end
14         1'd1: begin
15             pc_next = jump_addr_;
16         end
17     endcase
18 end
19 always_ff @(posedge clk) begin
20     if (rst|rst_pc_) begin
21         pc <= 0;
22     end
23     else begin
24         pc <= pc_next;
25     end
26 end
27
28 logic [31:0] inst_;
29 // program rom
30 Program_Rom myprogram_rom (
31     // in
32     .Rom_addr (pc),
33     // out
34     .Rom_data (inst_)
35 );
36
```

```

37     // pipe 1 IF_ID
38     logic flush_IFID_r, flush_IFID_;
39     logic [31:0] inst_r;
40     assign rst_or_flush_IFID_r = rst | flush_IFID_r;
41     always_ff @(posedge clk) begin
42         if (rst_or_flush_IFID_r) begin
43             inst_r <= `I_NOP;
44             pc_r <= 0;
45         end
46         else begin
47             inst_r <= inst_;
48             pc_r <= pc;
49         end
50     end
51
52     // INST_DEC
53     logic [6:0] funct7_, opcode_;
54     logic [4:0] addr_rs1_, addr_rs2_, addr_rd_;
55     logic [2:0] funct3_, funct3_r;
56     logic [31:0] imm_;
57     INST_DEC myinst_dec (
58         .inst_r (inst_r),
59         .funct7_ (funct7_),
60         .opcode_ (opcode_),
61         .addr_rs1_ (addr_rs1_),
62         .addr_rs2_ (addr_rs2_),
63         .addr_rd_ (addr_rd_),
64         .funct3_ (funct3_),
65         .imm_ (imm_)
66     );
67
68     // Reg file
69     logic write_regf_en_r, sel_rs1_value_, sel_rs2_value_;
70     logic [4:0] addr_rd_r;
71     logic [31:0] rd_value_, rs1_value, rs2_value, rs1_value_, rs2_value_, alu_out;
72     Reg_file myreg (
73         .clk (clk),
74         .rst (rst),
75         .write_regf_en (write_regf_en_r),
76         .addr_rd (addr_rd_r),
77         .addr_rs1 (addr_rs1_),
78         .addr_rs2 (addr_rs2_),
79         .rd_value (rd_value_),
80         .rs1_value (rs1_value),
81         .rs2_value (rs2_value),
82         .regs_31 (regs_31)
83     );
84     logic flush_INDEX_;

```

```

91     controller mycontroller(
92         // in
93         .funct7_(funct7_),
94         .funct3_(funct3_),
95         .opcode_(opcode_),
96         .rst_(rst),
97         .clk_(clk),
98         .rs1_value_(rs1_value_),
99         .rs2_value_(rs2_value_),
100        // out
101        .sel_alu_b_(sel_alu_b_),
102        .write_regf_en_(write_regf_en_),
103        .flush_IFID_(flush_IFID_),
104        .flush_INDEX_(flush_INDEX_),
105        .rst_pc_(rst_pc_),
106        .sel_pc_(sel_pc_),
107        .op_(op_),
108        .sel_jump_(sel_jump_),
109        .sel_alu_a_(sel_alu_a_),
110        .write_ram_(write_ram_),
111        .sel_rd_value_(sel_rd_value_)
112    );
113
114    // forwarding
115    assign sel_rs1_value_ = write_regf_en_r & (addr_rd_r == addr_rs1_);
116    assign sel_rs2_value_ = write_regf_en_r & (addr_rd_r == addr_rs2_);
117    assign rs1_value_ = sel_rs1_value_ ? rd_value_ : rs1_value;
118    assign rs2_value_ = sel_rs2_value_ ? rd_value_ : rs2_value;

```

```

120    // pipe 2 ID_EX
121    logic [31:0] imm_r, rs1_value_r, rs2_value_r, alu_b_, alu_a_, flush_INDEX_r, base_addr_, j_addr_, mul_out, div_out;
122    assign rst_or_flush_INDEX_r = rst | flush_INDEX_r;
123    always_ff @(posedge clk) begin
124        if (rst_or_flush_INDEX_r) begin
125            write_regf_en_r <= 0;
126            addr_rd_r <= 0;
127            imm_r <= 0;
128            rs1_value_r <= 0;
129            rs2_value_r <= 0;
130            op_r <= 0;
131            sel_alu_b_r <= 0;
132            pc_rr <= 0;
133            flush_INDEX_r <= 0;
134            flush_IFID_r <= 0;
135            sel_pc_r <= 0;
136            sel_jump_r <= 0;
137            sel_alu_a_r <= 0;
138            funct3_r <= 0;
139            write_ram_r <= 0;
140            sel_rd_value_r <= 0;
141        end
142        else begin
143            write_regf_en_r <= write_regf_en_;
144            addr_rd_r <= addr_rd_;
145            imm_r <= imm_;
146            rs1_value_r <= rs1_value_;
147            rs2_value_r <= rs2_value_;
148            op_r <= op_;
149            sel_alu_b_r <= sel_alu_b_;
150            pc_rr <= pc_r;
151            flush_INDEX_r <= flush_INDEX_;
152            flush_IFID_r <= flush_IFID_;
153            sel_pc_r <= sel_pc_;
154            sel_jump_r <= sel_jump_;
155            sel_alu_a_r <= sel_alu_a_;
156            funct3_r <= funct3_;
157            write_ram_r <= write_ram_;
158            sel_rd_value_r <= sel_rd_value_;
159        end
160    end

```

```

162      // multi 2 to 1 (sel_alu_a_r)
163      always_comb begin
164          case (sel_alu_a_r)
165              1'd0: alu_a_ = rs1_value_r;
166              1'd1: alu_a_ = pc_rr;
167          endcase
168      end
169
170      // multi 3 to 1 (sel_alu_b_r)
171      always_comb begin
172          unique case (sel_alu_b_r)
173              2'd0: alu_b_ = imm_r;
174              2'd1: alu_b_ = rs2_value_r;
175              2'd2: alu_b_ = 4;
176          endcase
177      end
178
179      // alu
180      myalu alu_1 (
181          .op (op_r),
182          .alu_a (alu_a_),
183          .alu_b (alu_b_),
184          .alu_out (alu_out)
185      );
186
187      // MUL myMul (
188      MUL myMul (
189          .funct3 (funct3_r),
190          .rs1_value (rs1_value_r),
191          .rs2_value (rs2_value_r),
192          .mul_out (mul_out)
193      );
194
195      // DIV
196      DIV myDiv (
197          .funct3 (funct3_r),
198          .rs1_value (rs1_value_r),
199          .rs2_value (rs2_value_r),
200          .div_out (div_out)
201      );

```

```
202 // LSU
203 logic [31:0] read_data;
204 mylsu lsu1 (
205     .clk (clk),
206     .write_ram (write_ram_r),
207     .funct3 (funct3_r),
208     .write_data (rs2_value_r),
209     .ram_addr (alu_out),
210     .read_data (read_data)
211 );
212
213
214 // multi 4 to 1
215 always_comb begin
216     case (sel_rd_value_r)
217         2'd0: rd_value_ = alu_out;
218         2'd1: rd_value_ = read_data;
219         2'd2: rd_value_ = mul_out;
220         2'd3: rd_value_ = div_out;
221     endcase
222 end
223
224 // jump_addr_ (adder)
225 // 2 to 1 multi (sel_jump_r)
226 assign base_addr_ = sel_jump_r ? pc_rr : rs1_value_r;
227 assign j_addr_ = base_addr_ + imm_r;
228 assign jump_addr_ = {j_addr_[31:1], (j_addr_[0]&sel_jump_r)};
229 endmodule
```

## myDefine.sv

```
my@mydevel:~/Desktop$ cat lnop.sv
1 `define I_NOP 32'h13
2
3 `define Opcode_I 7'b0010011
4 `define Opcode_R_M 7'b0110011
5 `define Opcode_B 7'b1100011
6 `define Opcode_L 7'b0000011
7 `define Opcode_S 7'b0010011
8
9 `define Opcode_JAL 7'b1101111
10 `define Opcode_JALR 7'b1100111
11 `define Opcode_LUI 7'b0110111
12 `define Opcode_AUIPC 7'b0010111
13
14 // alu operation
15 `define ALUOP_ADD 4'h0
16 `define ALUOP_SUB 4'h1
17 `define ALUOP_AND 4'h2
18 `define ALUOP_OR 4'h3
19 `define ALUOP_XOR 4'h4
20 `define ALUOP_A 4'h5
21 `define ALUOP_A_ADD_4 4'h6
22 `define ALUOP_LTU 4'h7
23 `define ALUOP_LT 4'h8
24 `define ALUOP_SLL 4'h9
25 `define ALUOP_SRL 4'hA
26 `define ALUOP_SRA 4'hB
27 `define ALUOP_B 4'hC
28
29 // function 3
30 `define F_ADDI 3'b000
31 `define F_SLTI 3'b010
32 `define F_SLTIU 3'b011
33 `define F_XORI 3'b100
34 `define F_ORI 3'b110
35 `define F_ANDI 3'b111
36 `define F_SLLI 3'b001
37 `define F_SRLI_SRAI 3'b101
38
39 // function 7
40 `define F7_ADD 7'b0000000
41 `define F7_SUB 7'b0100000
42 `define F7_SRLI 7'b0000000
43 `define F7_SRAI 7'b0100000
44 `define F7_OPCODE_R 7'b0000000
45 `define F7_SRL 7'b0000000
46 `define F7_SRA 7'b0100000
47 `define F7_R_M 7'b0000001
48
49 // alu
50 `define F_ADD_SUB 3'b000
51 `define F_SLL 3'b001
52 `define F_SLT 3'b010
53 `define F_SLTU 3'b011
54 `define F_XOR 3'b100
55 `define F_SRL_SRA 3'b101
56 `define F_OR 3'b110
57 `define F_AND 3'b111
58
59 // branch
60 `define F_BEQ 3'b000
61 `define F_BNE 3'b001
62 `define F_BLT 3'b100
63 `define F_BGE 3'b101
64 `define F_BLTU 3'b110
65 `define F_BGEU 3'b111
66
67 // LSU load
68 `define F_LB 3'b000
69 `define F_LH 3'b001
70 `define F_LW 3'b010
71 `define F_LBU 3'b100
72 `define F_LHU 3'b101
73 // store
74 `define F_SB 3'b000
75 `define F_SH 3'b001
76 `define F_SW 3'b010
```

```
78 // div, mux
79 `define F_MUL 3'b000
80 `define F_MULH 3'b001
81 `define F_MULHSU 3'b010
82 `define F_MULHU 3'b011
83 `define F_DIV 3'b100
84 `define F_DIVU 3'b101
85 `define F_Rem 3'b110
86 `define F_Remu 3'b111
```

## Mylsu.sv

```

④ myluv>_
1  `include "mydefine.sv"
2  `timescale 1ns/100ps
3  module mylsu(
4      input  logic clk,
5      input  logic write_ram,
6      input  logic [2:0] funct3,
7      input  logic [31:0] write_data,
8      input  logic [31:0] ram_addr,
9
10     output logic [31:0] read_data
11 );
12
13     logic [29:0] ram_addr_0;
14     logic [29:0] ram_addr_1;
15     logic [29:0] ram_addr_2;
16     logic [29:0] ram_addr_3;
17
18     logic [29:0] ram_addr_p4;
19
20     logic [7:0] read_data_0;
21     logic [7:0] read_data_1;
22     logic [7:0] read_data_2;
23     logic [7:0] read_data_3;
24
25     logic [7:0] write_data_0;
26     logic [7:0] write_data_1;
27     logic [7:0] write_data_2;
28     logic [7:0] write_data_3;
29
30     logic en_bank_0;
31     logic en_bank_1;
32     logic en_bank_2;
33     logic en_bank_3;
34
35     logic write_ram_0;
36     logic write_ram_1;
37     logic write_ram_2;
38     logic write_ram_3;
39
40     assign ram_addr_p4 = ram_addr[31:2] + 1;
41
42 //分配四塊 RAM 之位址和寫入資料
43 always_comb begin
44     unique case (ram_addr[1:0])
45         2'b00: begin
46             ram_addr_0 = ram_addr[31:2];
47             ram_addr_1 = ram_addr[31:2];
48             ram_addr_2 = ram_addr[31:2];
49             ram_addr_3 = ram_addr[31:2];
50             write_data_0 = write_data[7:0];
51             write_data_1 = write_data[15:8];
52             write_data_2 = write_data[23:16];
53             write_data_3 = write_data[31:24];
54         end
55         2'b01: begin
56             ram_addr_0 = ram_addr_p4;
57             ram_addr_1 = ram_addr[31:2];
58             ram_addr_2 = ram_addr[31:2];
59             ram_addr_3 = ram_addr[31:2];
60             write_data_0 = write_data[31:24];
61             write_data_1 = write_data[7:0];
62             write_data_2 = write_data[15:8];
63             write_data_3 = write_data[23:16];
64         end
65         2'b10: begin
66             ram_addr_0 = ram_addr_p4;
67             ram_addr_1 = ram_addr_p4;
68             ram_addr_2 = ram_addr[31:2];
69             ram_addr_3 = ram_addr[31:2];
70             write_data_0 = write_data[23:16];
71             write_data_1 = write_data[31:24];
72             write_data_2 = write_data[7:0];
73             write_data_3 = write_data[15:8];
74         end

```

```

75 v           2'bill: begin
76             ram_addr_0 = ram_addr_p4;
77             ram_addr_1 = ram_addr_p4;
78             ram_addr_2 = ram_addr_p4;
79             ram_addr_3 = ram_addr[31:2];
80             write_data_0 = write_data[15:8];
81             write_data_1 = write_data[23:16];
82             write_data_2 = write_data[31:24];
83             write_data_3 = write_data[7:0];
84         end
85     endcase
86 end
87
88 assign write_ram_0 = en_bank_0 & write_ram;
89 assign write_ram_1 = en_bank_1 & write_ram;
90 assign write_ram_2 = en_bank_2 & write_ram;
91 assign write_ram_3 = en_bank_3 & write_ram;
92
93
94 //决定哪些 RAM 可以写入
95 v           always_comb begin
96             en_bank_0 = 0;
97             en_bank_1 = 0;
98             en_bank_2 = 0;
99             en_bank_3 = 0;
100 v          unique case (funct3)
101 v            `F_SB: begin
102 v              unique case (ram_addr[1:0])
103                 2'b00: en_bank_0 = 1;
104                 2'b01: en_bank_1 = 1;
105                 2'b10: en_bank_2 = 1;
106                 2'b11: en_bank_3 = 1;
107             endcase
108         end
109 v            `F_SH: begin
110 v              unique case (ram_addr[1:0])
111                 2'b00: begin
112                     en_bank_0 = 1;
113                     en_bank_1 = 1;
114                 end
115 v                 2'b01: begin
116                     en_bank_1 = 1;
117                     en_bank_2 = 1;
118                 end
119 v                 2'b10: begin
120                     en_bank_2 = 1;
121                     en_bank_3 = 1;
122                 end
123 v                 2'b11: begin
124                     en_bank_3 = 1;
125                     en_bank_0 = 1;
126                 end
127             endcase
128         end
129 v            `F_SW: begin
130             en_bank_0 = 1;
131             en_bank_1 = 1;
132             en_bank_2 = 1;
133             en_bank_3 = 1;
134         end
135     endcase
136 end
137
138 v          RAM ram_0 (
139             .clk        (clk),
140             .write      (write_ram_0),
141             .write_data (write_data_0),
142             .ram_addr   (ram_addr_0),
143             .read_data  (read_data_0)
144         );

```

```

146
147     RAM ram_1 (
148         .clk          (clk),
149         .write        (write_ram_1),
150         .write_data   (write_data_1),
151         .ram_addr    (ram_addr_1),
152
153         .read_data   (read_data_1)
154     );
155
156     RAM ram_2 (
157         .clk          (clk),
158         .write        (write_ram_2),
159         .write_data   (write_data_2),
160         .ram_addr    (ram_addr_2),
161
162         .read_data   (read_data_2)
163     );
164
165     RAM ram_3 (
166         .clk          (clk),
167         .write        (write_ram_3),
168         .write_data   (write_data_3),
169         .ram_addr    (ram_addr_3),
170
171         .read_data   (read_data_3)
172     );
173
174     //組合四塊 RAM 的 read_data
175     always_comb begin
176         unique case (funct3)
177             'F_LB: begin
178                 unique case (ram_addr[1:8])
179                     2'b00: read_data = {{24{read_data_0[7]}}, read_data_0};
180                     2'b01: read_data = {{24{read_data_1[7]}}, read_data_1};
181                     2'b10: read_data = {{24{read_data_2[7]}}, read_data_2};
182                     2'b11: read_data = {{24{read_data_3[7]}}, read_data_3};
183                 endcase
184             end
185             'F_LH: begin
186                 unique case (ram_addr[1:8])
187                     2'b00: read_data = {{16{read_data_1[7]}}, read_data_1, read_data_0};
188                     2'b01: read_data = {{16{read_data_2[7]}}, read_data_2, read_data_1};
189                     2'b10: read_data = {{16{read_data_3[7]}}, read_data_3, read_data_2};
190                     2'b11: read_data = {{16{read_data_0[7]}}, read_data_0, read_data_3};
191                 endcase
192             end
193             'F_LW: begin
194                 unique case (ram_addr[1:8])
195                     2'b00: read_data = {read_data_3, read_data_2, read_data_1, read_data_0};
196                     2'b01: read_data = {read_data_0, read_data_3, read_data_2, read_data_1};
197                     2'b10: read_data = {read_data_1, read_data_0, read_data_3, read_data_2};
198                     2'b11: read_data = {read_data_2, read_data_1, read_data_0, read_data_3};
199                 endcase
200             end
201             'F_LBU: begin
202                 unique case (ram_addr[1:8])
203                     2'b00: read_data = {24'h0, read_data_0};
204                     2'b01: read_data = {24'h0, read_data_1};
205                     2'b10: read_data = {24'h0, read_data_2};
206                     2'b11: read_data = {24'h0, read_data_3};
207                 endcase
208             end
209             'F_LHU: begin
210                 unique case (ram_addr[1:8])
211                     2'b00: read_data = {16'h0, read_data_1, read_data_0};
212                     2'b01: read_data = {16'h0, read_data_2, read_data_1};
213                     2'b10: read_data = {16'h0, read_data_3, read_data_2};
214                     2'b11: read_data = {16'h0, read_data_0, read_data_3};
215                 endcase
216             end
217         endcase
218     end
219 endmodule

```

## Ram.sv

```

④ RAM.sv > ...
1 `timescale 1ns/100ps
2 module RAM (
3     input logic clk,
4     input logic write,
5     input logic [7:0] write_data,
6     input logic [29:0] ram_addr,
7     output logic [7:0] read_data
8 );
9     logic [7:0] ram[0:127];
10    assign read_data = ram[ram_addr];
11    always_ff @(posedge clk) begin
12        if (write) begin
13            ram[ram_addr] <= #1 write_data;
14        end
15    end
16 endmodule

```

## Reg\_file.sv

```

④ Reg_file.sv > ...
1 `timescale 1ns/100ps
2 module Reg_file(
3     input logic clk,
4     input logic rst,
5     input logic write_regf_en,
6     input logic [4:0] addr_rd,
7     input logic [4:0] addr_rsi,
8     input logic [4:0] addr_rs2,
9     input logic [31:0] rd_value,
10
11    output logic [31:0] rsi_value,
12    output logic [31:0] rs2_value,
13    output logic [31:0] regs_31
14 );
15
16    logic [31:0] regs[0:31];
17    logic addr_rd_not_0;
18    integer i;
19
20    assign regs_31 = regs[31];
21
22    assign addr_rd_not_0 = ~addr_rd;
23
24    assign rsi_value = regs[addr_rsi];
25    assign rs2_value = regs[addr_rs2];
26
27    always_ff@(posedge clk)
28    begin
29        if(rst) begin
30            for(i = 0; i < 32; i = i+1) begin:rst_keywords
31                regs[i] <= 0;
32            end
33        end
34        else begin
35            // Write
36            if (write_regf_en && addr_rd_not_0)
37                regs[addr_rd] <= #1 rd_value;
38        end
39    end
40
41 endmodule

```

**assembly**

```
asm > [s] code.s
1  init:
2  |    li x2, 0x341CDF0C
3  |    sw x2, 0(x0)
4  |    li x2, 0xBD006365
5  |    sw x2, 4(x0)
6  |    li x2, 0xF
7  |    sb x2, 8(x0)
8  |    li x3, 0x9
9  |    li x2, 0xE055FF5
10 |    sw x2, 0(x3)
11 |    li x2, 0x34434DF7
12 |    sw x2, 4(x3)
13 |    li x2, 0xC
14 |    sb x2, 8(x3)
15
16 #-----開始撰寫您的組合語言-----
17 addi x2, x0, 0      # init x2 = 0, load matrix A index
18 addi x6, x0, 0x9    # init x6 = 0x9, load matrix B index
19 addi x12, x0, 0x12  # init memory start multiple ans address
20 add x13, x0, x0     # index of loopOne
21 add x14, x0, x0     # index of loopSecond
22 addi x15, x0, 3     # bound of loop end
23 loopOne:
24    lb x3, 0(x2)      # source A index[0]
25    lb x4, 1(x2)      # source A index[1]
26    lb x5, 2(x2)      # source A index[2]
27    addi x2, x2, 3     # shift A index + 3
28    j loopSecond
29
30 loopSecond:
31    lb x7, 0(x6)      # source B index[0]
32    lb x8, 3(x6)      # source B index[1]
33    lb x9, 6(x6)      # source B index[2]
34    addi x6, x6, 1     # shift B index + 1
```

```
35      # try to multiple
36      add x10, x0, x0      # store ans of multiple
37      mul x11, x3, x7      # A[0]*B[0]
38      add x10, x10, x11
39      mul x11, x4, x8      # A[1]*B[1]
40      add x10, x10, x11
41      mul x11, x5, x9      # A[2]*B[2]
42      add x10, x10, x11
43
44
45      # try to store ans
46      sh x10, 0(x12)      # store ans into memory
47      addi x12, x12, 2      # shift memory 2 bytes
48      addi x14, x14, 1      # add one of loop2
49      j loopSecondCmp
50
51  loopSecondCmp:
52      bne x14, x15, loopSecond
53      add x14, x0, x0      # init index of loop2
54      addi x6, x0, 0x9      # init x6, index of source B
55      addi x13, x13, 1      # add one of loop1
56  loopOneCmp:
57      bne x13, x15, loopOne
58      add x2, x0, x0      # init x2, index of source A
59
60  #-----您的組合語言-----
61
62      li x3, 0x12
63      li x4, 9
64  done:
65      lh x31, 0(x3)
66      addi x3, x3, 2
67      addi x4, x4, -1
68      bne x4, x0, done
```

## Program\_rom.sv

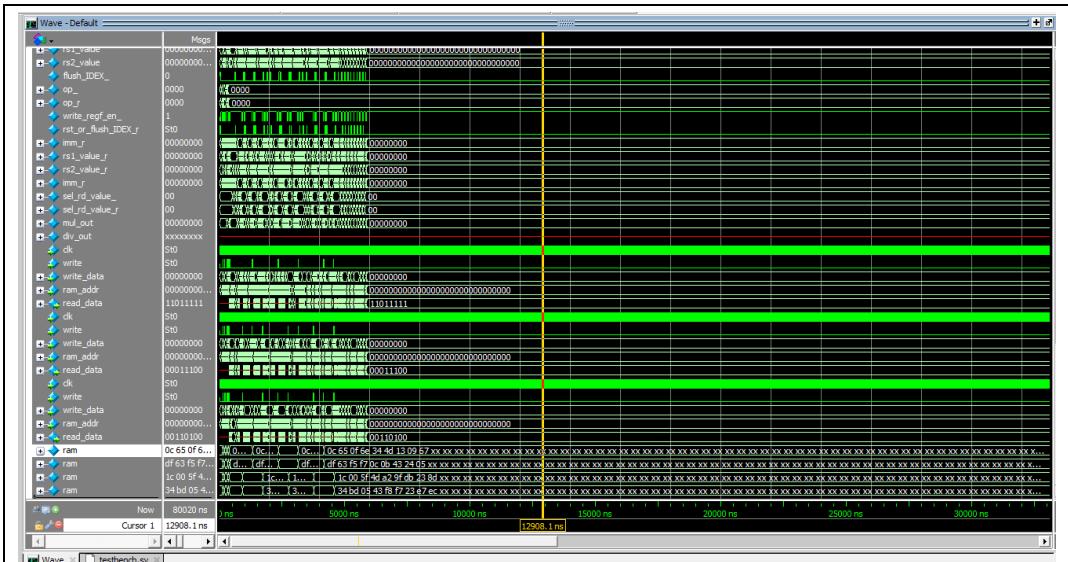
```
Program_Rom.sv > Program_Rom
1 module Program_Rom(
2     output logic [31:0] Rom_data,
3     input [31:0] Rom_addr
4 );
5
6     always_comb begin
7         case (Rom_addr)
8             32'h00000000 : Rom_data = 32'h341CE137; // lui x2 213454
9             32'h00000004 : Rom_data = 32'hF0C10113; // addi x2 x2 -244
10            32'h00000008 : Rom_data = 32'h00202023; // sw x2 0(x0)
11            32'h0000000C : Rom_data = 32'hBD006137; // lui x2 774150
12            32'h00000010 : Rom_data = 32'h36510113; // addi x2 x2 869
13            32'h00000014 : Rom_data = 32'h00202223; // sw x2 4(x0)
14            32'h00000018 : Rom_data = 32'h00F00113; // addi x2 x0 15
15            32'h0000001C : Rom_data = 32'h00200423; // sb x2 8(x0)
16            32'h00000020 : Rom_data = 32'h00900193; // addi x3 x0 9
17            32'h00000024 : Rom_data = 32'h6E056137; // lui x2 450646
18            32'h00000028 : Rom_data = 32'hFF510113; // addi x2 x2 -11
19            32'h0000002C : Rom_data = 32'h0021A023; // sw x2 0(x3)
20            32'h00000030 : Rom_data = 32'h34435137; // lui x2 214069
21            32'h00000034 : Rom_data = 32'hDF710113; // addi x2 x2 -521
22            32'h00000038 : Rom_data = 32'h0021A223; // sw x2 4(x3)
23            32'h0000003C : Rom_data = 32'h00C00113; // addi x2 x0 12
24            32'h00000040 : Rom_data = 32'h00218423; // sb x2 8(x3)
25            32'h00000044 : Rom_data = 32'h00000113; // addi x2 x0 0
26            32'h00000048 : Rom_data = 32'h00900313; // addi x6 x0 9
27            32'h0000004C : Rom_data = 32'h01200613; // addi x12 x0 18
28            32'h00000050 : Rom_data = 32'h000006B3; // add x13 x0 x0
29            32'h00000054 : Rom_data = 32'h00000733; // add x14 x0 x0
30            32'h00000058 : Rom_data = 32'h00300793; // addi x15 x0 3
31            32'h0000005C : Rom_data = 32'h00010183; // lb x3 0(x2)
32            32'h00000060 : Rom_data = 32'h00110203; // lb x4 1(x2)
33            32'h00000064 : Rom_data = 32'h00210283; // lb x5 2(x2)
34            32'h00000068 : Rom_data = 32'h00310113; // addi x2 x2 3
35            32'h0000006C : Rom_data = 32'h0040006F; // jal x0 4
36            32'h00000070 : Rom_data = 32'h00030383; // lb x7 0(x6)
37            32'h00000074 : Rom_data = 32'h00330403; // lb x8 3(x6)
38            32'h00000078 : Rom_data = 32'h00630483; // lb x9 6(x6)
39            32'h0000007C : Rom_data = 32'h00130313; // addi x6 x6 1
40            32'h00000080 : Rom_data = 32'h00000533; // add x10 x0 x0
41            32'h00000084 : Rom_data = 32'h027185B3; // mul x11 x3 x7
42            32'h00000088 : Rom_data = 32'h00B50533; // add x10 x10 x11
43            32'h0000008C : Rom_data = 32'h028205B3; // mul x11 x4 x8
44            32'h00000090 : Rom_data = 32'h00B50533; // add x10 x10 x11
45            32'h00000094 : Rom_data = 32'h029285B3; // mul x11 x5 x9
46            32'h00000098 : Rom_data = 32'h00B50533; // add x10 x10 x11
```

```

47      32'h0000000C : Rom_data = 32'h00A61023; // sh x10 0(x12)
48      32'h000000A0 : Rom_data = 32'h00260613; // addi x12 x12 2
49      32'h000000A4 : Rom_data = 32'h00170713; // addi x14 x14 1
50      32'h000000A8 : Rom_data = 32'h0040006F; // jal x0 4
51      32'h000000AC : Rom_data = 32'hFCF712E3; // bne x14 x15 -60
52      32'h000000B0 : Rom_data = 32'h00000733; // add x14 x0 x0
53      32'h000000B4 : Rom_data = 32'h00900313; // addi x6 x0 9
54      32'h000000B8 : Rom_data = 32'h00168693; // addi x13 x13 1
55      32'h000000BC : Rom_data = 32'hFAF690E3; // bne x13 x15 -96
56      32'h000000C0 : Rom_data = 32'h00000133; // add x2 x0 x0
57      32'h000000C4 : Rom_data = 32'h01200193; // addi x3 x0 18
58      32'h000000C8 : Rom_data = 32'h00900213; // addi x4 x0 9
59      32'h000000CC : Rom_data = 32'h00019F83; // lh x31 0(x3)
60      32'h000000D0 : Rom_data = 32'h00218193; // addi x3 x3 2
61      32'h000000D4 : Rom_data = 32'hFFF20213; // addi x4 x4 -1
62      32'h000000D8 : Rom_data = 32'hFE021AE3; // bne x4 x0 -12
63      default : Rom_data = 32'h00000013;      // NOP
64      endcase
65  end
66
67 endmodule

```

## ● 模擬結果與結果說明：



可以看到下方四個 ram，與提供的解答相同。

## ● 結論與心得：

終於有前饋了，寫 assembly 終於不用再考慮該死的 nop 了，真

是太棒了：）