



2025/12/04

實驗十二

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_HW12

1、組語撰寫練習—64 位元乘法

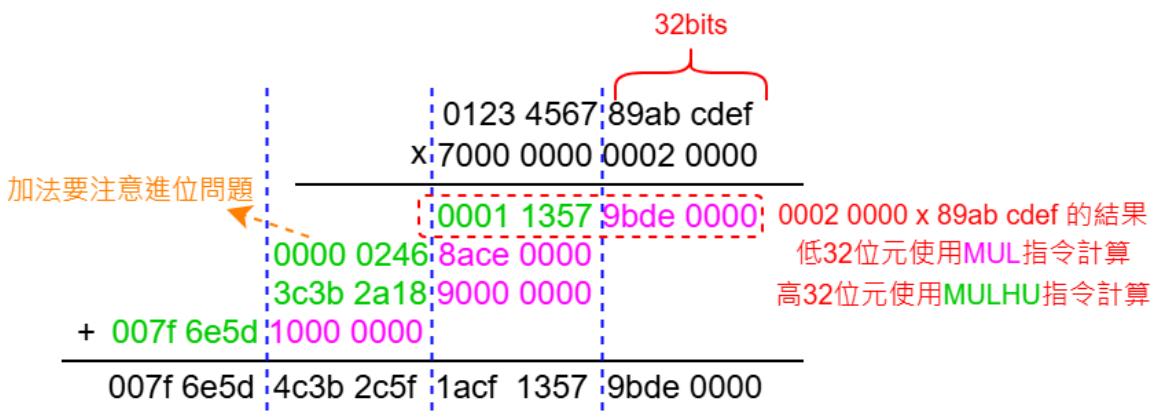
● 實驗說明：

1. 請透過當前課堂所學的組合語言撰寫一個 64 位元的有號數乘法，計算 $0x0123_4567_89ab_cdef$ 乘以 $0x7000_0000_0002_0000$ 的結果，如下圖一所示。程式碼須加註解。

$$\begin{aligned} 0x1234_567_89ab_cdef \times 0x7000_0000_0002_0000 \\ = 0x007f_6e5d_4c3b_2c5f_1acf_1357_9bde_0000 \end{aligned}$$

2. 64 位元的被乘數與乘數分別使用兩個暫存器儲存。
3. 在相乘前須先判斷被乘數與乘數是否為負數，若是則先取二補數。
(Hint：較高的 32bits 取反向，較低的 32bits 反向加一，並判斷是否須進位)
4. 有無進位可以使用圖二中的組合語言來判斷，若'和'小於加數與被加數則表示有進位發生。

5. 最終計算出來的結果要依據乘數與被乘數的符號來決定是否要取二補數。
6. 所有暫存器皆可使用，但請將結果位置由小到大載入到 **x31 ~ x28** 暫存器中，如 **0x9bde_0000** 存入 **x31**。
7. 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。
8. 請將所有用到的暫存器都加入至波形中以方便觀察。
9. 請注意資料危障的問題。



圖一

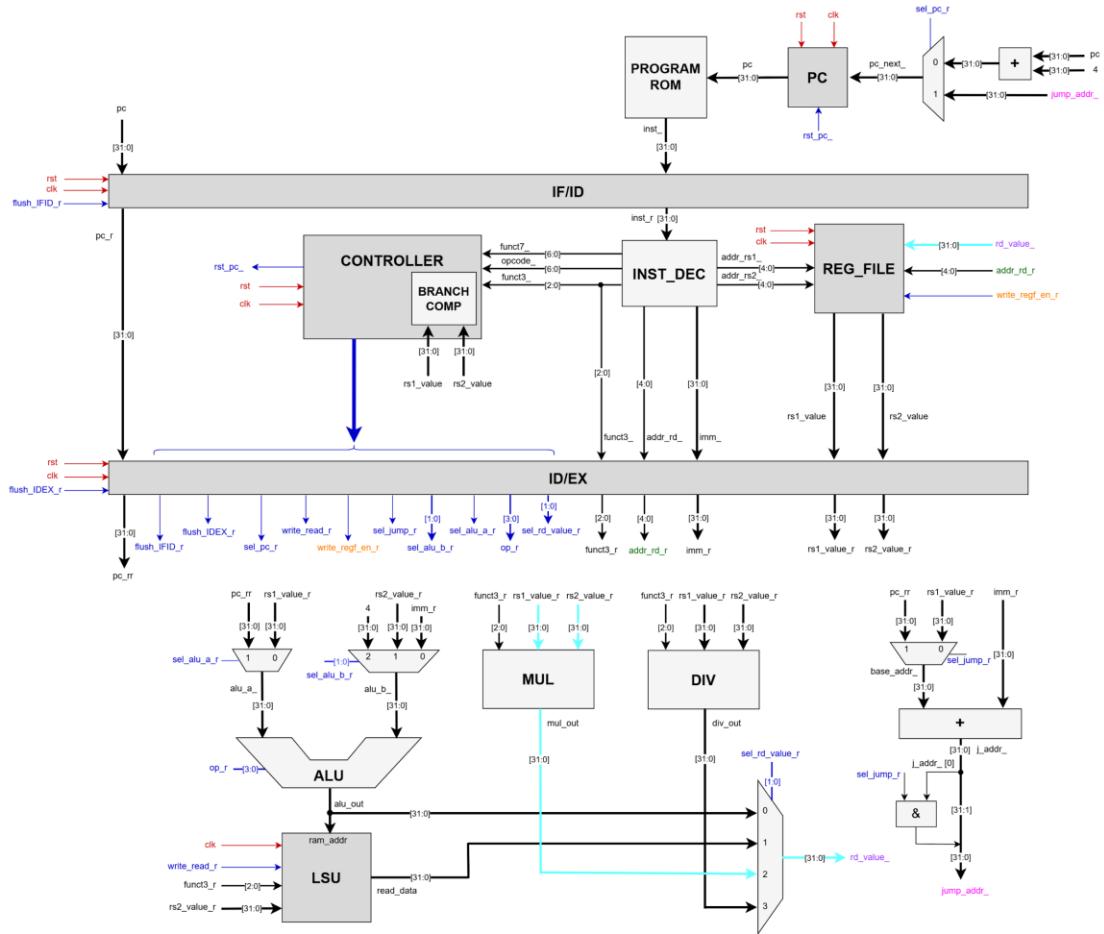
```

1 # x1, x2: 32 位元加數與被加數
2 # x5, x4: 64 位元存放加法結果，x4存低32位元的結果，x5存高32位元的結果
3 # x3: 用來存放進位標誌 (1 表示有進位，0 表示無進位)
4
5 add x4, x1, x2          # x4 = x1 + x2 (可能有進位)
6 sltu x3, x4, x1         # 如果 x4 < x1，則有進位 (x3 = 1)，否則無進位 (x3 = 0)
7 add x5, x5, x3          # x5 = x5 + x3 加上進位

```

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

```
combsimav> d 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
13  logic BEQ_FLAG;
14  logic BNE_FLAG;
15  logic BLT_FLAG;
16  logic BGE_FLAG;
17  logic BLTU_FLAG;
18  logic BGEU_FLAG;
19
20  assign BEQ_FLAG = (rs1_value == rs2_value);
21  assign BNE_FLAG = (rs1_value != rs2_value);
22  assign BLT_FLAG = ($signed(rs1_value) < $signed(rs2_value));
23  assign BGE_FLAG = ($signed(rs1_value) >= $signed(rs2_value));
24  assign BLTU_FLAG = (rs1_value < rs2_value);
25  assign BGEU_FLAG = (rs1_value >= rs2_value);
26
27  typedef enum {s0, s1, s2} fsm_state;
28  fsm_state ps, ns;
29
30  always_ff @(posedge clk) begin
31      if (rst) begin
32          ps <= #1 s0;
33      end
34      else begin
35          ps <= #1 ns;
36      end
37  end
38
39  always_comb begin
40      rst_pc_ = 0;
41      sel_pc_ = 0;
42      flush_IFID_ = 0;
43      flush_INDEX_ = 0;
44      write_regf_en_ = 0;
45      sel_alu_b_ = 0;
46      ns = ps;
47      op_ = 'ALUOP_ADD;
48      sel_alu_a_ = 0;
49      sel_jump_ = 0;
50      sel_rd_value_ = 0;
51      write_ram_ = 0;
52      unique case (ps)
53          s0: begin
54              flush_IFID_ = 1;
55              flush_INDEX_ = 1;
56              rst_pc_ = 1;
57              ns = s1;
58          end
59          s1: begin
60              flush_IFID_ = 1;
61              flush_INDEX_ = 1;
62              rst_pc_ = 1;
63              ns = s2;
64          end
65      endcase
66  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
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
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
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
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
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
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999

```

```

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         end
255         `F_BLTU: begin
256             if (BLTU_FLAG) begin
257                 sel_pc_ = 1;
258                 flush_INDEX_ = 1;
259                 flush_IFID_ = 1;
260             end
261         end
262         `F_BGEU: begin
263             if (BGEU_FLAG) begin
264                 sel_pc_ = 1;
265                 flush_INDEX_ = 1;
266                 flush_IFID_ = 1;
267             end
268         end
269     endcase
270 end
271 `Opcode_JAL: begin
272     sel_pc_ = 1;
273     flush_IFID_ = 1;
274     flush_INDEX_ = 1;
275     sel_alu_a_ = 1;
276     sel_alu_b_ = 2;
277     sel_jump_ = 1;
278     op_ = `ALUOP_ADD;
279     write_regf_en_ = 1;
280 end
281 `Opcode_JALR: begin
282     sel_pc_ = 1;
283     flush_IFID_ = 1;
284     flush_INDEX_ = 1;
285     sel_alu_a_ = 1;
286     sel_alu_b_ = 2;
287     sel_jump_ = 0;
288     op_ = `ALUOP_ADD;
289     write_regf_en_ = 1;
290 end
291 `Opcode_LUI: begin
292     sel_alu_b_ = 0;
293     op_ = `ALUOP_B;
294     write_regf_en_ = 1;
295 end
296 `Opcode_AUIPC: begin
297     sel_alu_a_ = 1;
298     sel_alu_b_ = 0;
299     op_ = `ALUOP_ADD;
300     write_regf_en_ = 1;
301 end
302 `Opcode_I: begin
303     unique case (funct3_)
304         `F_LB: begin
305             op_ = `ALUOP_ADD;
306             sel_rd_value_ = 1;
307             write_regf_en_ = 1;
308         end
309         `F_LH: begin
310             op_ = `ALUOP_ADD;
311             sel_rd_value_ = 1;
312             write_regf_en_ = 1;
313         end
314         `F_LW: begin
315             op_ = `ALUOP_ADD;
316             sel_rd_value_ = 1;
317             write_regf_en_ = 1;
318         end
319         `F_LBU: begin
320             op_ = `ALUOP_ADD;
321             sel_rd_value_ = 1;
322             write_regf_en_ = 1;
323         end
324         `F_LHU: begin
325             op_ = `ALUOP_ADD;
326             sel_rd_value_ = 1;
327             write_regf_en_ = 1;
328         end
329     endcase
330 end

```

```
332     Upcode_S: begin
333         unique case (funct3_)
334             'F_SB: begin
335                 op_ = `ALUOP_ADD;
336                 write_ram_ = 1;
337             end
338             'F_SH: begin
339                 op_ = `ALUOP_ADD;
340                 write_ram_ = 1;
341             end
342             'F_SW: begin
343                 op_ = `ALUOP_ADD;
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 endmodule
19
```

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

```

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_i;
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;
70      logic [4:0] addr_rd_r;
71      logic [31:0] rd_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_;
85    // controller
86    logic sel_pc_, sel_alu_a_r, sel_jump_, sel_jump_r, sel_alu_a_, write_ram_, write_ram_r;
87    logic [1:8] sel_rd_value_, sel_rd_value_r;
88    logic [1:8] sel_alu_b_r, sel_alu_b_;
89    logic [3:8] op_, op_r;
90    logic write_regf_en_;
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 // pipe 2 ID_EX
115 logic [31:8] imm_r, rs1_value_r, rs2_value_r, alu_b_, alu_a_, flush_INDEX_r, base_addr_, j_addr_, mul_out, div_out;
116 assign rst_or_flush_INDEX_r = rst | flush_INDEX_r;
117 always_ff @(posedge clk) begin
118     if (rst_or_flush_INDEX_r) begin
119         write_regf_en_r <= 0;
120         addr_rd_r <= 0;
121         imm_r <= 0;
122         rs1_value_r <= 0;
123         rs2_value_r <= 0;
124         op_r <= 0;
125         sel_alu_b_r <= 0;
126         pc_rr <= 0;
127         flush_INDEX_r <= 0;
128         flush_IFID_r <= 0;
129         sel_pc_r <= 0;
130         sel_jump_r <= 0;
131         sel_alu_a_r <= 0;
132         funct3_r <= 0;
133         write_ram_r <= 0;
134         sel_rd_value_r <= 0;
135     end
136     else begin
137         write_regf_en_r <= write_regf_en_;
138         addr_rd_r <= addr_rd_;
139         imm_r <= imm;
140         rs1_value_r <= rs1_value;
141         rs2_value_r <= rs2_value;
142         op_r <= op_;
143         sel_alu_b_r <= sel_alu_b_;
144         pc_rr <= pc_r;
145         flush_INDEX_r <= flush_INDEX_;
146         flush_IFID_r <= flush_IFID_;
147         sel_pc_r <= sel_pc_j;
148         sel_jump_r <= sel_jump_;
149         sel_alu_a_r <= sel_alu_a_;
150         funct3_r <= funct3_j;
151         write_ram_r <= write_ram_;
152         sel_rd_value_r <= sel_rd_value_;
153     end
154 end

```

```

156 // multi 2 to 1 (sel_alu_a_r)
157 always_comb begin
158     case (sel_alu_a_r)
159         1'd0: alu_a_ = rs1_value_r;
160         1'd1: alu_a_ = pc_rr;
161     endcase
162 end
163
164 // multi 3 to 1 (sel_alu_b_r)
165 always_comb begin
166     unique case (sel_alu_b_r)
167         2'd0: alu_b_ = imm_r;
168         2'd1: alu_b_ = rs2_value_r;
169         2'd2: alu_b_ = 4;
170     endcase
171 end
172
173 // alu
174 myalu alu_1 (
175     .op (op_r),
176     .alu_a (alu_a_),
177     .alu_b (alu_b_),
178     .alu_out (alu_out)
179 );
180
181 // mul
182 MUL myMul (
183     .funct3 (funct3_r),
184     .rs1_value (rs1_value_r),
185     .rs2_value (rs2_value_r),
186     .mul_out (mul_out)
187 );
188
189 // div
190 DIV myDiv (
191     .funct3 (funct3_r),
192     .rs1_value (rs1_value_r),
193     .rs2_value (rs2_value_r),
194     .div_out (div_out)
195 );
196
197 // LSU
198 logic [31:0] read_data;
199 mylsu lsu1 (
200     .clk (clk),
201     .write_ram (write_ram_r),
202     .funct3 (funct3_r),
203     .write_data (rs2_value_r),
204     .ram_addr (alu_out),
205     .read_data (read_data)
206 );
207
208 // multi 4 to 1
209 always_comb begin
210     case (sel_rd_value_r)
211         2'd0: rd_value_ = alu_out;
212         2'd1: rd_value_ = read_data;
213         2'd2: rd_value_ = mul_out;
214         2'd3: rd_value_ = div_out;
215     endcase
216 end
217
218 // jump_addr_ (adder)
219 // 2 to 1 multi (sel_jump_r)
220 assign base_addr_ = sel_jump_r ? pc_rr : rs1_value_r;
221 assign j_addr_ = base_addr_ + imm_r;
222 assign jump_addr_ = {j_addr_[31:1], (j_addr_[0]&sel_jump_r)};
223
endmodule

```

myDefine.sv

```
mydefines.v 7 UNDEF
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'b0100011
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

```
① 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 //分配四块 RAM 之位址和写入资料
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     end
76 end
```

```

75      2'b11: 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 always_comb begin
96     en_bank_0 = 0;
97     en_bank_1 = 0;
98     en_bank_2 = 0;
99     en_bank_3 = 0;
100    unique case (funct3)
101        'F_SB: begin
102            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        'F_SH: begin
110            unique case (ram_addr[1:0])
111                2'b00: begin
112                    en_bank_0 = 1;
113                    en_bank_1 = 1;
114                end
115                2'b01: begin
116                    en_bank_1 = 1;
117                    en_bank_2 = 1;
118                end
119                2'b10: begin
120                    en_bank_2 = 1;
121                    en_bank_3 = 1;
122                end
123                2'b11: begin
124                    en_bank_3 = 1;
125                    en_bank_0 = 1;
126                end
127            endcase
128        end
129        '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    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_IW: 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

```

1 `timescale ins/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

```

1 `timescale ins/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] rs1_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 rs1_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

```

Program_rom + assembly

$$\begin{array}{c} X_2, X_3 \\ X \quad X_4, X_5 \\ \hline X_6, X_7 \\ X_8, X_9 \\ X_{10}, X_{11} \\ X_{12}, X_{13} \\ \hline X_{28} \quad X_{29} \quad X_{30} \quad X_{31} \\ \uparrow \quad \uparrow \quad \uparrow \\ X_{14} \quad X_{15} \quad X_{16} \end{array}$$

$X_{17} = 1$
 $X_{18} = \text{flag}$

```
asm > [3] codes.s
1  init:
2      # li x2, 0x01234567
3      lui x2, 0x01234
4      nop
5      addi x2, x2, 0x567
6
7      # li x3, 0x89abcdef
8      lui x3, 0x89abd
9      nop
10     addi x3, x3, -0x211
11
12     # li x4, 0x70000000
13     lui x4, 0x70000
14     nop
15     addi x4, x4, 0x000
16
17     # li x5, 0x00020000
18     lui x5, 0x00020
19     nop
20     addi x5, x5, 0x000
21
22     # used to move eight byte into reg (y * 1 = 1)
23     addi x17, x0, 1
24
25     # Each state of multiple
26     mulhu x6, x5, x3
27     mul x7, x5, x3
28
29     mulhu x8, x5, x2
30     mul x9, x5, x2
31
32     mulhu x10, x4, x3
33     mul x11, x4, x3
34
35     mulhu x12, x4, x2
36     mul x13, x4, x2
```

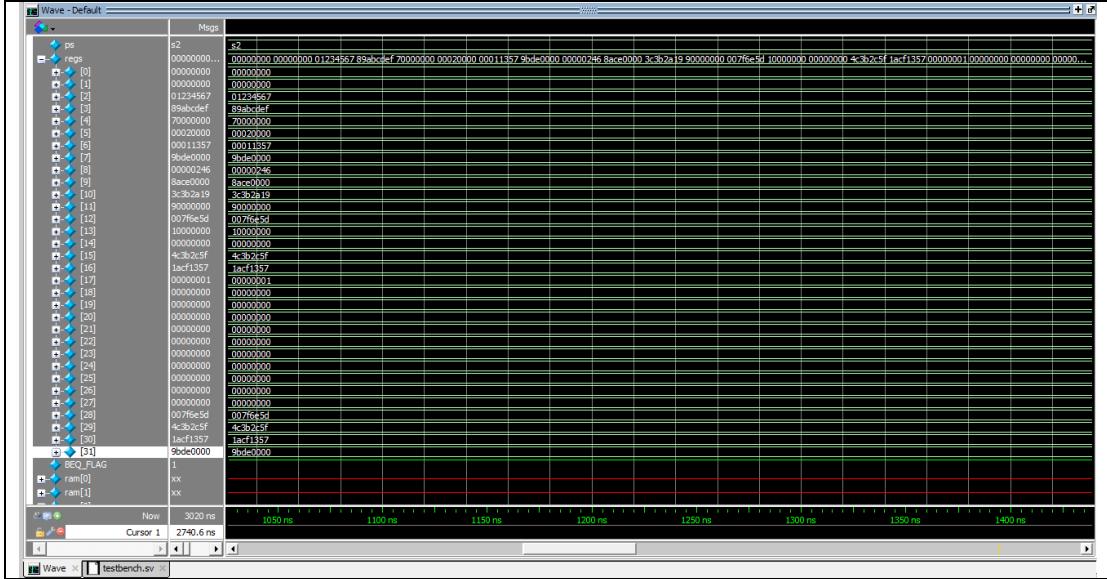
```
37
38     # move x7 => x13
39     mul x31, x7, x17
40
41     # x16 = x6 + x9 + x11 (nop for x16 and x18)
42     add x16, x6, x9
43     nop
44     sltu x18, x16, x9
45     nop
46     add x8, x8, x18
47
48     add x16, x16, x11
49     nop
50     sltu x18, x16, x11
51     nop
52     add x10, x10, x18
53
54     # move x16 => x30
55     mul x30, x16, x17
56
57     # x15 = x8 + x10 + x13
58     add x15, x8, x10
59     nop
60     sltu x18, x15, x10
61     nop
62     add x12, x12, x18
63
64     add x15, x15, x13
65     nop
66     sltu x18, x15, x13
67     nop
68     add x12, x12, x18
69
70     # move x15 => x29
71     mul x29, x15, x17
72
73     # move x12 => x28
74     mul x28, x12, x17
75     nop
```

```

@: Program_Rom.sv > Program_Rom
1  module Program_Rom(
2    output logic [31:0] Rom_data,
3    input [31:0] Rom_addr
4  );
5    always_comb begin
6      case (Rom_addr)
7        32'h00000000 : Rom_data = 32'h01234137; // lui x2 4660
8        32'h00000004 : Rom_data = 32'h00000013; // addi x0 x0 0
9        32'h00000008 : Rom_data = 32'h56710113; // addi x2 x2 1383
10       32'h0000000C : Rom_data = 32'h89ABD1B7; // lui x3 563901
11       32'h00000010 : Rom_data = 32'h00000013; // addi x0 x0 0
12       32'h00000014 : Rom_data = 32'hDEF18193; // addi x3 x3 -529
13       32'h00000018 : Rom_data = 32'h70000237; // lui x4 458752
14       32'h0000001C : Rom_data = 32'h00000013; // addi x0 x0 0
15       32'h00000020 : Rom_data = 32'h00020213; // addi x4 x4 0
16       32'h00000024 : Rom_data = 32'h00020287; // lui x5 32
17       32'h00000028 : Rom_data = 32'h00000013; // addi x0 x0 0
18       32'h0000002C : Rom_data = 32'h00028293; // addi x5 x5 0
19       32'h00000030 : Rom_data = 32'h00100893; // addi x17 x0 1
20       32'h00000034 : Rom_data = 32'h0232B333; // mulhu x6 x5 x3
21       32'h00000038 : Rom_data = 32'h0232B3B3; // mul x7 x5 x3
22       32'h0000003C : Rom_data = 32'h0222B433; // mulhu x8 x5 x2
23       32'h00000040 : Rom_data = 32'h0222B4B3; // mul x9 x5 x2
24       32'h00000044 : Rom_data = 32'h02323533; // mulhu x10 x4 x3
25       32'h00000048 : Rom_data = 32'h023205B3; // mul x11 x4 x3
26       32'h0000004C : Rom_data = 32'h02223633; // mulhu x12 x4 x2
27       32'h00000050 : Rom_data = 32'h022206B3; // mul x13 x4 x2
28       32'h00000054 : Rom_data = 32'h03138FB3; // mul x31 x7 x17
29       32'h00000058 : Rom_data = 32'h00930833; // add x16 x6 x9
30       32'h0000005C : Rom_data = 32'h00000013; // addi x0 x0 0
31       32'h00000060 : Rom_data = 32'h00983933; // sltu x18 x16 x9
32       32'h00000064 : Rom_data = 32'h00000013; // addi x0 x0 0
33       32'h00000068 : Rom_data = 32'h01240433; // add x8 x8 x18
34       32'h0000006C : Rom_data = 32'h00B80833; // add x16 x16 x11
35       32'h00000070 : Rom_data = 32'h00000013; // addi x0 x0 0
36       32'h00000074 : Rom_data = 32'h00B83933; // sltu x18 x16 x11
37       32'h00000078 : Rom_data = 32'h00000013; // addi x0 x0 0
38       32'h0000007C : Rom_data = 32'h01250533; // add x10 x10 x18
39       32'h00000080 : Rom_data = 32'h03180F33; // mul x30 x16 x17
40       32'h00000084 : Rom_data = 32'h00A407B3; // add x15 x8 x10
41       32'h00000088 : Rom_data = 32'h00000013; // addi x0 x0 0
42       32'h0000008C : Rom_data = 32'h00A7B933; // sltu x18 x15 x10
43       32'h00000090 : Rom_data = 32'h00000013; // addi x0 x0 0
44       32'h00000094 : Rom_data = 32'h01260633; // add x12 x12 x18
45       32'h00000098 : Rom_data = 32'h00D787B3; // add x15 x15 x13
46       32'h0000009C : Rom_data = 32'h00000013; // addi x0 x0 0
47       32'h000000A0 : Rom_data = 32'h00D7B933; // sltu x18 x15 x13
48       32'h000000A4 : Rom_data = 32'h00000013; // addi x0 x0 0
49       32'h000000A8 : Rom_data = 32'h01260633; // add x12 x12 x18
50       32'h000000AC : Rom_data = 32'h03178EB3; // mul x29 x15 x17
51       32'h000000B0 : Rom_data = 32'h03160E33; // mul x28 x12 x17
52       32'h000000B4 : Rom_data = 32'h00000013; // addi x0 x0 0
53       default : Rom_data = 32'h00000013; // NOP
54     endcase
55   end
56 endmodule

```

●模擬結果與結果說明：



●結論與心得：

我知道乘法組語不是那麼簡單，但我不想再往後檢查，向前一位檢察溢位已經是我的極限了，我不想一直考慮到最開頭的地方了，沒有每次那麼剛好從做後加一到開頭一路進位的：）