

Single Cycle Datapath Processor using MIPS

Piero Morales

Computer Science University Student
University of Engineering and Technology
Lima, Peru
piero.morales@utec.edu.pe

Angel Motta

Computer Science University Student
University of Engineering and Technology
Lima, Peru
angel.motta@utec.edu.pe

Abstract—The form, design, and implementation of CPUs have changed over the course of their history, but their fundamental operation remains almost unchanged. The CPU has become the nerve center of any computer, from mobile devices to supercomputers. From the beginning of computer era scientists have tried to improve processor performance not only increasing the number of transistors, but also by improving the instructions that the processor executes. A major change that happened for CPUs is the change from single core to multi core that increasing they performance. In this way, Moore's law, that until this moment had traced the future of processors, is discarded.

Index Terms—Computer architecture, risc, verilog, processor, big endian, microprocessor without interlocked pipeline stages

I. INTRODUCTION

There are two major architectures for CPUs, the Reduced Instruction Set Computer (RISC) and Complex Instruction Set Computer (CISC). RISC design architecture points to solve problems about the CPU time processing although compare with CISC it increase the lines of code to the software developer, nevertheless the main advantage of using RISC architecture is the reduced amount of clock cycles needed to executed and instruction due to the specialized instructions.

In this context we have the 32 bits MIPS Instruction Set Architecture (ISA) [1] which support all the necessary functions needed by the software, MIPS is composed by 32 general-purpose registers and instructions format to classify all the instructions. There are 3 types of instruction formats: the R-Type which uses 3 registers, the I-Type who uses 2 registers and a 16-bit immediate value and finally we have the J-Type who supports the "jumps" between the lines of instructions.

There is no definite architecture in the present CPU technology, the industry use both RISC and CISC architecture separately or in combination, which of those are going to use will depend on the requirement.

II. METHODOLOGY

Since manufacturing a physical processor require a state-of-the-art technology and also a huge amount of money, we use an Hardware Description Language (HDL) to design and simulate our processor and all the components related. We choose Verilog [2] as HDL because is widely used in the industry and the access to the student license for ModelSim [3].

We choose the single cycle as a design methodology with focussing in the basic operations with integers, covering the

following R-type, I-type and J-type instructions from the 32 bits MIPS ISA:

TABLE I
R TYPE

Instructions		
ADD	Subtraction (SUB)	AND
NOR	OR	Set Less Than (SLT)
Jump Register (JR)		

TABLE II
I TYPE

Instructions		
Add Immediate (ADDI)	Subtraction Immediate (SUBI)	AND Immediate (ANDI)
OR Immediate (ORI)	Set Less Than Immediate (SLTI)	Store Byte (SB)
Store Halfword (SH)	Store Word (SW)	Load Byte (LB)
Load Halfword (LH)	Load Word (LW)	Load Upper Immediate (LUI)
Branch On Equal (BEQ)	Branch On Not Equal (BNEQ)	Branch On Greater than equal zero (BGEZ)

TABLE III
J TYPE

Instructions		
Jump (J)	Jump and Link (JAL)	

A. Datapath

In order to cover all the instructions mentioned we need to model the following components:

- Instruction Memory, stores the instructions to be executed.
- PC Counter, points to the line of the instruction of the program which not necessary always will be the following next since the we are using branches and jumps.
- Register File, stores the 32 registers for the MIPS ISA.
- Data Memory, stores the data for the program and also the stack.

- Arithmetic Logic Unit (ALU), the "brain" of the processor who make the operations of addition, subtraction, the comparison between two numbers, logic AND, logic OR, logic NOR.
- Multiplexor 2 to 1, this component indicates which of the 2 inputs input take based on a selector signal i.e. in the selection between the PC Counter, the branch or the jump.
- Adder, we use this to add the number 4 to the actual PC counter to point to the next instruction, also is used for the offset to cover the branch instruction.
- Shift Left 2 and 16, to be used to calculate the offset for the branch and load a number up to 32 bits respectively.
- Sign extend, this component is used to extend the most significant bit of the number.
- and the control component for support all the instructions deciding which signal activate depending on the type of instruction and the operation.

Putting all the components together we get our datapath:

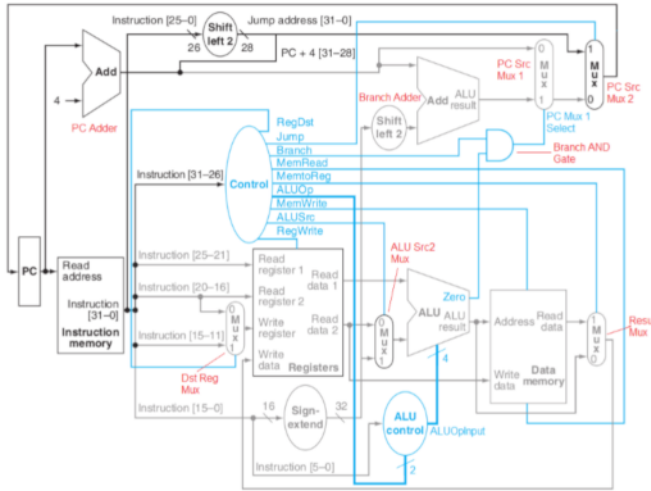


Fig. 1. Datapath.

As we can see all the components are connected using wires.

B. ModelSim

We start developing our components in verilog, in order to maintenance and following the standard principles of software developing we create each module separately in its corresponding .v file, e.g. regfile.v for the Register File component. In total we have the following 17 files for all the modules:

```
regfile_lab5.v      shift16_module.v
alu_lab5.v          instrumem.v
extendbit_lab5.v   ControlDataPath.v
alucontrol.v        mux31.v
Select_word_half.v mux_5bits.v
shift_left_2.v      pccounter_4.v
PC_module.v         concatenar.v
adder32.v           complete_32bits.v
proyecto_final.v
```

To clarify the terms every CPU component will be created in ModelSim as a module [4].

```
C:/Programas/Modeltech_pe_edu_10.4a/examples/Sabado/regfile_lab5.v - Default
Ln#
1  //-----
2  // Design Name : regfile
3  // File Name   : regfile_lab5.v
4  // Function    : Store the MIPS 32 registers
5  // Coder       : Raúl Mosquera Pumarićra
6  //-----
7  module regfile(in1, in2, in3, wreg, clk, sel, out1, out2);
8      input in1;
9      input in2;
10     input in3;
11     input clk;
12     input wreg;
13     input sel;
14     output out1;
15     output out2;
16
17     wire sel;
18     wire [4:0] in1, in2, in3;
19     wire [31:0] wreg;
20     wire [31:0] out1, out2;
21     reg[31:0] memory[31:0];
22
```

Fig. 2. Register File module in ModelSim.

III. EXPERIMENTAL SETUP

To test our datapath we first test each component separately to isolate the problems, once all the components pass its respective tests we continue with a complete test of our datapath, in order to do that we agrupar the previous instructions in 3 files to be loaded in the instruction memory.

Each test bench file contains the instructions and its respective operands (registers). Previously for purpose testing we load the register file with random values.

TABLE IV
REGISTERS

Register number	Number in decimal notation	Number in hexadecimal notation
16	49527	0000C177
17	63767	0000F917
18	31778	00007C22
19	23198	00005A9E
20	917	00000395
21	24182	00005E76
22	52687	0000CDCF
23	20726	000050F6
29	150	00000096

The others registers, 1 to 15, 24 to 28 and 30,31 remain with zero.

In each test bench are using 10 nanoseconds as a positive clock signal and negative clock signal which give us 20 nanoseconds in total per clock cycle.

To get a real approach of the use of the processor we will run the following C code.

The translation into MIPS instructions it would be as follow:

As we can use the factorial function will use most of the intructions implemented and the recursivity technique, this will be our test bench 4.

TABLE V
TESTBENCH 1

Instructions		
ADD	Subtraction (SUB)	AND
NOR	OR	Set Less Than (SLT)
Add Immediate (ADDI)	Subtraction Immediate (SUBI)	AND Immediate (ANDI)
OR Immediate (ORI)	Set Less Than Immediate (SLTI)	

TABLE VI
TESTBENCH 2

Instructions		
Store Byte (SB)	Store Halfword (SH)	Store Word (SW)
Load Byte (LB)	Load Halfword (LH)	Load Word (LW)
Load Upper Immediate (LUI)		

To calculate the CPU time [5] (time processing) for each test bench we use the following formula:

$$Time = PI * CPI * TimeperClockCycle$$

where PI is Program Instructions and CPI is Clock Cycles per instruction.

IV. EVALUATION

We executed the test bench 1, 2, 3 and 4, these were the results:

We ran the test bench in intervals of 100 nanoseconds as we can see in Fig. 5. . For the test bench 1 we get 100 nanoseconds + 100 nanoseconds + 20 nanoseconds with in total give us 220 nanoseconds.

We apply the same procedure to the test bench 2 , Fig. 8. and the result was 100 nanoseconds + 40 nanoseconds = 140 nanoseconds.

In test bench 3, Fig. 9., we got 100 nanoseconds + 100 nanoseconds + 100 nanoseconds + 40 nanoseconds with th total of 340 nanoseconds.

And finally for the test bench 4, Fig. 6, we use intervals of 500 nanoseconds since the execution is elevated, we get 2610 nanoseconds in total also the output of the factorial of 10 was 0x003750f00 which in decimal notation correspond to 3628800.

Comparing the results of Fig. 5., Fig. 8. and Fig. 9. with the Table VIII we get the same amount of clock cycles and

TABLE VII
TESTBENCH 3

Instructions		
Branch On Equal (BEQ)	Branch On Not Equal (BNEQ)	Branch On Greater than equal zero (BGEZ)
Jump (J)	Jump and Link (JAL)	Jump Register (JR)

Since we need to simulate the branch and the jumps between the instructions additional instructions were added i.e. ADDI and SUB

```
int fact(int n){
    if (n<1)
        return 1;
    else
        return n*factorial(n-1)
}
variable = factorial(10);
```

Fig. 3. Factorial function - C code.

ADDI	\$a0	\$0	10
JAL		factorial	
ADD	\$s0	\$v0	\$0
SUBI	\$sp	\$sp	8
SW	\$a0	\$sp	0
SW	\$ra	\$sp	4
SLTI	\$t0	\$a0	1
BEQ	\$t0	\$0	label1
ADDI	\$v0	\$0	1
ADDI	\$sp	\$sp	8
JR		\$ra	
SUBI	\$a0	\$a0	1
JAL		factorial	
LW	\$a0	\$sp	0
LW	\$ra	\$sp	4
ADDI	\$sp	\$sp	8
MULTI	\$v0	\$v0	\$a0
JR		\$ra	

Fig. 4. Factorial function - MIPS.

the time for each file, also the results of the instructions are as we expected.

V. CONCLUSION

- Since the single cycle datapath was the original solution in the early days of RISC architecture nowadays is not efficient because it takes 1 cycle per instruction when another aproachs using the pipeline technique could reduce the cycle needed per instruction.
- It became mandatory calculate the expected time processing for test our datapath, otherwise we could get unexpected result since the execution continue with the next instruction, like for example in our test bench for the factorial.
- Before start coding in ModelSim we need to decide if we are going to apply the clock edge triggered and which modules will apply for that, because if not the major changes needed in the modules would be very risky and will take more time for testing.
- Verilog was not specially designed to upload files that why for our tests purposes we need to modify the code to point a specific file.

VI. COMMENTS

- When we are simulating our component in ModelSim no warnings must appear when the simulation starts,

TABLE VIII
CLOCK CYCLES

	Total instructions	Total executed instructions (Expected)	Clock Cycles	CPU Time (Nanoseconds)
Test bench 1	11	11	11	220
Test bench 2	7	7	7	140
Test bench 3	22	17	17	340
Test bench 4	18	131	131	2620

```

1 module finalpy_testbench_1;
2   reg[48:0] str_op;
3   reg clk;
4
5   instrument.v   FinalTestBench1.v
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 Operation = SUB Clock= 0 Resultado = 0x00007cd9 Overflow = 0
21 Operation = AND Clock= 1 Resultado = 0x00000c17 Overflow = 0
22 Operation = AND Clock= 0 Resultado = 0x00000c17 Overflow = 0
23 Operation = NOR Clock= 1 Resultado = 0xfffff819 Overflow = 0
24 Operation = NOR Clock= 0 Resultado = 0xfffff819 Overflow = 0
25 Operation = OR Clock= 1 Resultado = 0x00007ebe Overflow = 0
26 Operation = OR Clock= 0 Resultado = 0x00007ebe Overflow = 0
27 Operation = SLT Clock= 1 Resultado = 0x00000000 Overflow = 0
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100 Operation = SLT Clock= 0 Resultado = 0x00000000 Overflow = 0
101 Operation = ADD Clock= 1 Resultado = 0x000005da7 Overflow = 0
102 Operation = ADD Clock= 0 Resultado = 0x000005da7 Overflow = 0
103 Operation = SUB Clock= 1 Resultado = 0x00000c08b Overflow = 0
104 Operation = SUB Clock= 0 Resultado = 0x00000c08b Overflow = 0
105 Operation = AND Clock= 1 Resultado = 0x0000010f6 Overflow = 0
106 Operation = AND Clock= 0 Resultado = 0x0000010f6 Overflow = 0
107 Operation = OR Clock= 1 Resultado = 0x000007fff Overflow = 0
108 Operation = OR Clock= 0 Resultado = 0x000007fff Overflow = 0
109 Operation = SLT Clock= 1 Resultado = 0x00000000 Overflow = 0
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200 Operation = SLT Clock= 0 Resultado = 0x00000000 Overflow = 0
201 Operation = SLT Clock= 1 Resultado = 0xffffffff Overflow = x
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
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
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Fig. 5. Execution results for test bench 1.

otherwise there was some error or unexpected behaviour. One common issue is referring a wire or register as an input or output of a module with different length.

- In ModelSim is the identifier is not declare verilog assume is a wire.
- The verilog compiler doesn't warn you when a module instantiation does not exists until you simulate it
- One common problem is asume the execution of the code in the components of the datapath will be sequential, that is not correct since we have the always @ block and that could be executed in the upper sign of the clock or the lower sign.
- For those who are used to the conditional statements of the programming languages it is a little difficult at the beginning use verilog, because at the digital circuit level there we only have and, or, xor and all the gates.
- To find the errors in the testing fase we can navigate in the windows objects in ModelSim throught the modules to find the issue.

REFERENCES

- [1] MIPS.com. (2016). MIPS® Architecture for Programmers Volume II-A: The MIPS32® Instruction Set Manual. [online] Available at: <https://s3-eu-west-1.amazonaws.com/downloads-mips/documents/MD00086-2B-MIPS32BIS-AFP-6.06.pdf> [Accessed 27 Nov. 2018].
- [2] IEEE Standard for Verilog Hardware Description Language. IEEE Standard 1364-2005 (Revision of IEEE Standard 1364-2001). <http://dx.doi.org/10.1109/IEEESTD.2006.99495>, 2006. Last access 26 November 2018.

```

1 module finalpy_testbench_1;
2   reg[48:0] str_op;
3   reg clk;
4
5   instrument.v   FinalTestBench1.v
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 Operation = SUB Clock= 0 Resultado = 0x00007cd9 Overflow = 0
21 Operation = AND Clock= 1 Resultado = 0x00000c17 Overflow = 0
22 Operation = AND Clock= 0 Resultado = 0x00000c17 Overflow = 0
23 Operation = NOR Clock= 1 Resultado = 0xfffff819 Overflow = 0
24 Operation = NOR Clock= 0 Resultado = 0xfffff819 Overflow = 0
25 Operation = OR Clock= 1 Resultado = 0x00007ebe Overflow = 0
26 Operation = OR Clock= 0 Resultado = 0x00007ebe Overflow = 0
27 Operation = SLT Clock= 1 Resultado = 0x00000000 Overflow = 0
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100 Operation = SLT Clock= 0 Resultado = 0x00000000 Overflow = 0
101 Operation = ADD Clock= 1 Resultado = 0x000005da7 Overflow = 0
102 Operation = ADD Clock= 0 Resultado = 0x000005da7 Overflow = 0
103 Operation = SUB Clock= 1 Resultado = 0x00000c08b Overflow = 0
104 Operation = SUB Clock= 0 Resultado = 0x00000c08b Overflow = 0
105 Operation = AND Clock= 1 Resultado = 0x0000010f6 Overflow = 0
106 Operation = AND Clock= 0 Resultado = 0x0000010f6 Overflow = 0
107 Operation = OR Clock= 1 Resultado = 0x000007fff Overflow = 0
108 Operation = OR Clock= 0 Resultado = 0x000007fff Overflow = 0
109 Operation = SLT Clock= 1 Resultado = 0x00000000 Overflow = 0
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200 Operation = SLT Clock= 0 Resultado = 0x00000000 Overflow = 0
201 Operation = SLT Clock= 1 Resultado = 0xffffffff Overflow = x
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
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
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Fig. 6. Execution results of the test bench for the factorial.

Input:

10!

Result:

3 628 800

Fig. 7. Wolfram Alpha - Factorial of 10. [6]

- [3] Mentor.com. (2018). ModelSim PE Student Edition. [online] Available at: https://www.mentor.com/company/higher_ed/modelsim-student-edition [Accessed 27 Nov. 2018].
- [4] Ashenden, P. (2008). Digital Design: An Embedded Systems Approach Using Verilog. Burlington, MA: Elsevier Science, pp.22,23.
- [5] Patterson, D., Hennessy, J. and Alexander, P. (2012). Computer organization and design. 4th ed. Waltham, Mass: Morgan Kaufmann, pp.35.
- [6] Wolframalpha.com (2018). Wolfram—Alpha: Making the world's knowledge computable. [online] Wolframalpha.com. Availableat:<https://www.wolframalpha.com/input/?i=factorial+10> [Accessed 28 Nov. 2018].

```

C:/Programas/Modeltech_pe_edu_10.4a/examples/ProyectoFinal/FinalTestBench2.v (/finalpj_testbench_2) - Default
Ln#
1 include "proyecto_final.v";
2
3 module finalpj_testbench_2;
4
5 //wire[31:0] instruction;

instrument.v x FinalTestBench2.v x
Transcript
V$IM 26> run
# 0 Operation = SB Clock= 0 Resultado = 0x000000f6 Overflow = 0
# 10 Operation = SH Clock= 1 Resultado = 0x000000395 Overflow = 0
# 20 Operation = SH Clock= 0 Resultado = 0x000000395 Overflow = 0
# 30 Operation = SW Clock= 1 Resultado = 0x0000f917 Overflow = 0
# 40 Operation = SW Clock= 0 Resultado = 0x0000f917 Overflow = 0
# 50 Operation = LB Clock= 1 Resultado = 0x000000f6 Overflow = 0
# 60 Operation = LB Clock= 0 Resultado = 0x000000f6 Overflow = 0
# 70 Operation = LH Clock= 1 Resultado = 0x000000395 Overflow = 0
# 80 Operation = LH Clock= 0 Resultado = 0x000000395 Overflow = 0
# 90 Operation = LW Clock= 1 Resultado = 0x0000f917 Overflow = 0
V$IM 27> run
# 100 Operation = LW Clock= 0 Resultado = 0x0000f917 Overflow = 0
# 110 Operation = LUI Clock= 1 Resultado = 0x06f10000 Overflow = 0
# 120 Operation = LUI Clock= 0 Resultado = 0x06f10000 Overflow = 0
# 130 Operation = LUI Clock= 1 Resultado = 0xxxxxxx Overflow = x

```

Fig. 8. Execution results for test bench 2.

```

C:/Programas/Modeltech_pe_edu_10.4a/examples/ProyectoFinal/FinalTestBench3.v (/finalpj_testbench_3) - Default
Ln#
127 clk = 0;
128 end
129 #10 begin
130 // JUMP
131 str on = "JR";

instrument.v x FinalTestBench3.v x
Transcript
# 100 Operation = BGEZ Clock= 0 Resultado = 0x00000020 Overflow = 0
# 110 Operation = ADDi Clock= 1 Resultado = 0x00000099 Overflow = 0
# 120 Operation = ADDi Clock= 0 Resultado = 0x00000099 Overflow = 0
# 130 Operation = ADDi Clock= 1 Resultado = 0x0000009a Overflow = 0
# 140 Operation = ADDi Clock= 0 Resultado = 0x0000009a Overflow = 0
# 150 Operation = ADDi Clock= 1 Resultado = 0x0000009b Overflow = 0
# 160 Operation = ADDi Clock= 0 Resultado = 0x0000009b Overflow = 0
# 170 Operation = JUMP Clock= 1 Resultado = 0x00000034 Overflow = 0
# 180 Operation = JUMP Clock= 0 Resultado = 0x00000034 Overflow = 0
# 190 Operation = SUBI Clock= 1 Resultado = 0x00000000 Overflow = 0
run
# 200 Operation = SUBI Clock= 0 Resultado = 0x00000000 Overflow = 0
# 210 Operation = JAL Clock= 1 Resultado = 0x00000050 Overflow = 0
# 220 Operation = JAL Clock= 0 Resultado = 0x00000050 Overflow = 0
# 230 Operation = ADDI Clock= 1 Resultado = 0x00000003 Overflow = 0
# 240 Operation = ADDI Clock= 0 Resultado = 0x00000003 Overflow = 0
# 250 Operation = JR Clock= 1 Resultado = 0x0000003c Overflow = 0
# 260 Operation = JR Clock= 0 Resultado = 0x0000003c Overflow = 0
# 270 Operation = ADDI Clock= 1 Resultado = 0x000000f4 Overflow = 0
# 280 Operation = ADDI Clock= 0 Resultado = 0x000000f4 Overflow = 0
# 290 Operation = JR Clock= 1 Resultado = 0x00000048 Overflow = 0
V$IM 21> run
# 300 Operation = JR Clock= 0 Resultado = 0x00000048 Overflow = 0
# 310 Operation = ADDI Clock= 1 Resultado = 0x000000f9 Overflow = 0
# 320 Operation = ADDI Clock= 0 Resultado = 0x000000f9 Overflow = 0
# 330 Operation = Clock= 1 Resultado = 0x00000000 Overflow = 0

```

Fig. 9. Execution results for test bench 3.