

UMBC CSEE

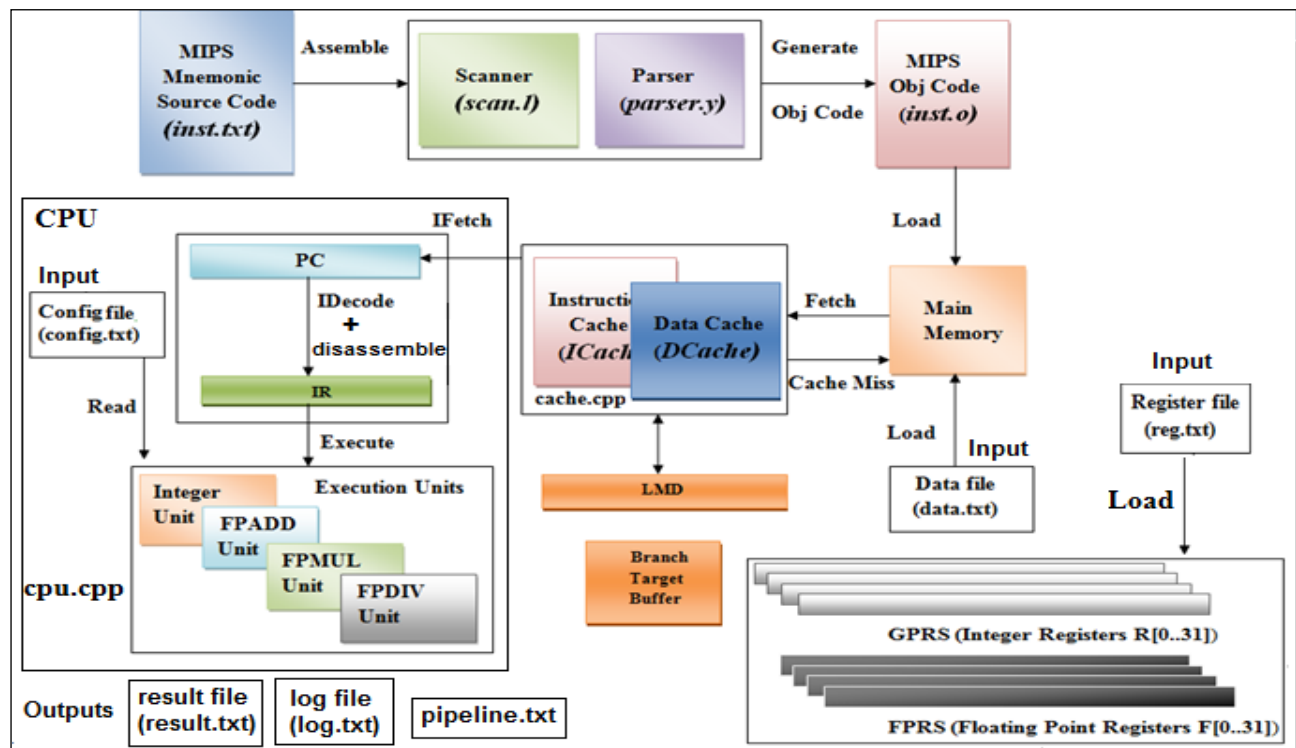
Project- MIPS Simulator

Advanced Computer Architecture

Sandipan Dey

12/19/2009

Design



Test Cases run & their outputs

1. [Project Statement Document \(Project_Fall2009.pdf\) Sample Test case](#)

(The one in the project statement using the same data.txt reg.txt config.txt as specified in the document with the following slight change in inst.txt)

inst.txt

```
L.D F6, 779(R2)      ; changing the offset from 34 to 779 since the target address must be in [256, 287]
L.D F2, 45(R3)       ; changing the offset slightly since the target address must be in [256, 287]
MUL.D F0, F2, F4
SUB.D F8, F6, F2
DIV.D F10, F0, F6
ADD.D F6, F8, F2
```

It generates the following output, which is **exactly identical** to one expected:

```
Disassembly      1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
LD F6, 779(R2)           IF ID EX M1 M2 M3 M4 M5 M6 WB
LD F2, 822(R3)           IF ID EX S S S S S M1 M2 M3 M4 M5 M6 WB
MUL.D F0, F2, F4         IF ID S S S S S S EX1 EX2 EX3 EX4 EX5 EX6 WB
SUB.D F8, F6, F2         IF ID S S S S S S EX1 EX2 EX3 EX4 WB
DIV.D F10, F0, F6        IF ID S S S S S S EX1 EX2 EX3 EX4 EX5 EX6 EX7 EX8 EX9 EX10
ADD.D F6, F8, F2         IF ID S S EX1 EX2 EX3 EX4 WB
```

```

linux1[16]% simulator inst.txt data.txt reg.txt config.txt result.txt

assembling source file inst1.txt...
generating object file inst1.o...

object code:
110111000100011000000001100001011
110111000110001000000001100110110
010001000000001000001000000000010
010001000000000100011001000000001
010001000000001100000001010000011
010001000000000100100000110000000

parsing the config file config1.txt...
parsing done!

loading the data file data1.txt in memory at address 0x100...
data file loaded!
loading the registers file reg1.txt...
registers file loaded!

loading the object file inst1.o at address 0x000 ...
object file successfully loaded...

starting execution...
finishing execution...

writing results to file result.txt...

LD F6, 779(R2)          6      7      14      15
LD F2, 822(R3)          7      8      20      21
MUL.D F0, F2, F4        13     14     26     27
SUB.D F8, F6, F2        14     15     24     25
DIV.D F10, F0, F6       20     21     46     47
ADD.D F6, F8, F2        21     22     28     29

Total number of access requests for instruction cache:      6
Number of instruction cache hits:          3
Total number of access requests for data cache:            2
Number of data cache hits:                  0

```

[2. Test case 5 \(CMSC611 Test Cases/Test 5/inst.txt\)](#)

```

L.D F0, 0(R0)
L.D F2, 64(R0)
L.D F4, 0(R0)
L.D F4, 120(R0)
L.D F6, 64(R0)
L.D F8, 0(R0)
L.D F6, 34(R2)
BEQ R1, R1, Loop
MUL.D F0, F2, F4
ADD.D F10, F4, F2
Loop: ADD.D F10, F2, F4
L.D F6, 34(R2)
MUL.D F0, F2, F4
ADD.D F10, F4, F2
DIV.D F2, F8, F2

```

Actual Output					Expected Output				
LD F0, 0(R0)	6	7	14	15	l.d f0, 0(r0)	6	7	14	15
LD F2, 64(R0)	12	13	20	21	l.d f2, 64(r0)	12	13	20	21
LD F4, 0(R0)	18	19	26	27	l.d f4, 0(r0)	18	19	26	27
LD F4, 120(R0)	24	26	33	34	l.d f4, 120(r0)	24	26	33	34
LD F6, 64(R0)	31	32	39	40	l.d f6, 64(r0)	30	31	39	40
LD F8, 0(R0)	37	38	45	46	l.d f8, 0(r0)	36	37	45	46
LD F6, 34(R2)	43	44	51	52	l.d f6, 34(r2)	42	43	51	52
BEQ R1, R1, I11	49	50			beq r1,r1,loop	48	49		
MUL.D F0, F2, F4	55				mul.d f0,f2,f4	54			
ADD.D F10, F2, F4	61	62	68	69	add.d f10,f2,f4	60	61	67	68
LD F6, 34(R2)	67	68	70	71	l.d f6, 34(r2)	66	67	70	71
MUL.D F0, F2, F4	73	74	80	81	mul.d f0, f2, f4	72	73	79	80
ADD.D F10, F4, F2	79	80	86	87	add.d f10, f4, f2	78	79	85	86
DIV.D F2, F8, F2	85	86	92	93	div.d f2, f8, f2	84	85	91	92
Total number of access requests for instruction cache: 14					Total number of access requests for instruction cache: 14				
Number of instruction cache hits: 0					Number of instruction cache hits:0				
Total number of access requests for data cache: 8					Total number of access requests for data cache: 8				
Number of data cache hits: 1					Number of data cache hits:1				

Disassembly	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
LD F0, 0(R0)						IF	ID	EX	M1	M2	M3	M4	M5	M6	WB																					
LD F2, 64(R0)												IF	ID	EX	M1	M2	M3	M4	M5	M6	WB															
LD F4, 0(R0)																	IF	ID	EX	M1	M2	M3	M4	M5	M6	WB										
LD F4, 120(R0)																								IF	S	ID	EX	M1	M2	M3	M4	M5	M6	WB		
LD F6, 64(R0)																															IF	ID	EX	M1	M2	M3
LD F8, 0(R0)																																				
LD F6, 34(R2)																																				
BEQ R1, R1, I11																																				
MUL.D F0, F2, F4																																				
ADD.D F10, F2, F4																																				
LD F6, 34(R2)																																				
MUL.D F0, F2, F4																																				
ADD.D F10, F4, F2																																				
DIV.D F2, F8, F2																																				

The only place where the actual output (**result v1.txt**) differs from expected output is the ID stage in the **4th instruction**. But since there is a **WAW hazard** detected in between the 3rd & 4th instructions (both having the same output register **F4**) at the beginning of the ID stage of the 4th instruction (25th cycle), the ID stage of 4th instruction can't complete (it can't enter the EX stage) before the 3rd one enters the WB stage. So it must stall at ID stage till 26th (the previous load completes fetching data at 26th cycle only).

Also, the stall is **before** ID stage (as can be seen from the above figures), because in my design, a stall at a particular stage means before that stage (since stall after ID will mean stall before EX, it stalls before, not after ID). Also, since IF is stalled if ID is stalled (they are sequential in nature) there will be this extra one latency introduced everywhere.

An instruction waiting in 1-cycle wide ID stage means that it has stalls either in the beginning or end of the ID stage (the cant wait inside the ID stage, since in practice it will be a D-type / T-type master-slave F/F, either the values will be stored in IF->ID latch (IF -> ID -> EX -> WB)

But having stall at the end of ID stage means that it must wait at the beginning of the EX stage, which is not the case here, that means it must wait at the beginning of it

[3. Test case 6 \(CMSC611 Test Cases/Test 6/inst.txt\)](#)

```

LOOP: LW R4, 256(R0)
      LW R5, 260(R0)
      MUL.D F0,F2,F4
      BEQ R1,R2,DONE

```

```

DADD R1,R1,R1
SW R5, 256(R0)
J LOOP
DONE: DIV.D F10,F0,F6
L.D F6, 1024(R2)
MUL.D F20,F22,F24
ADD.D F21,F24,F22
DIV.D F22,F28,F22

```

```

simulator inst.txt data.txt reg.txt config.txt result.txt
assembling source file inst.txt...
generating object file inst.o...

object code:
1000110000000010000000000100000000
1000110000000010100000000100000100
0100010000000010000001000000000010
00010000000100010000000000000001000
0000000000010000100001000001011100
10101100000001010000000100000000
0000100000000000000000000000000001
010001000000011000000001010000011
11011100010001100000010000000000
0100010000011000101010100000010

```

```

writing results to file result.txt...
LW R4, 256(R0)      4      5      10     11
LW R5, 260(R0)      8      9      14     15
MUL.D F0, F2, F4    12     13     19     20
BEQ R1, R2, I8      16     17
DADD R1, R1, R1     20     21     23     24
SW R5, 256(R0)     24     25     30
J I1                28     29
DIV.D F10, F0, F6   32
LW R4, 256(R0)     36     37     39     40
LW R5, 260(R0)     40     41     46     47
MUL.D F0, F2, F4    44     45     51     52
BEQ R1, R2, I8      48     49
DADD R1, R1, R1     52     53     55     56
SW R5, 256(R0)     56     57     62
J I1                60     61
DIV.D F10, F0, F6   64     65     67     68
LW R4, 256(R0)     68     69     74     75
LW R5, 260(R0)     72     73     79     80
MUL.D F0, F2, F4    76     77
BEQ R1, R2, I8      84     85     91     92
DADD R1, R1, R1     88     89     94     95
SW R5, 256(R0)     92     93     99    100
J I1                96     97    103    104
DIV.D F22, F28, F22 100    101    107    108

Total number of access requests for instruction cache: 25
Number of instruction cache hits: 0

Total number of access requests for data cache: 9
Number of data cache hits: 2

saved results to file result.txt...

```

For this test case the result is **exactly identical** to the one that is expected (result_v1.txt). Still it does not exactly match while comparing by diff for 2 reasons: 1) for difference in spaces, 2) During disassembling I **regenerate** the label names, but this time they are different (a label is named by **I10** if it corresponds to 10th instruction) 3) After disassembling, the source mnemonics my simulator generators are all in capital letters.

Expected Output					Actual Output				
lw r4,256(r0)	4	5	10	11	writing results to file result.txt...				
lw r5,260(r0)	8	9	14	15	LW R4, 256(R0)	4	5	10	11
mul.d f0,f2,f4	12	13	19	20	LW R5, 260(R0)	8	9	14	15
beq r1,r2,done	16	17			MUL.D F0, F2, F4	12	13	19	20
dadd r1,r1,r1	20	21	23	24	BEQ R1, R2, I8	16	17		
sw r5,256(r0)	24	25	30		DADD R1, R1, R1	20	21	23	24
j loop	28	29			SW R5, 256(R0)	24	25	30	
div.d f10,f0,f6	32				J I1	28	29		
lw r4,256(r0)	36	37	39	40	DIV.D F10, F0, F6	32			
lw r5,260(r0)	40	41	46	47	LW R4, 256(R0)	36	37	39	40
mul.d f0,f2,f4	44	45	51	52	LW R5, 260(R0)	40	41	46	47
beq r1,r2,done	48	49			MUL.D F0, F2, F4	44	45	51	52
dadd r1,r1,r1	52	53	55	56	BEQ R1, R2, I8	48	49		
sw r5,256(r0)	56	57	62		DADD R1, R1, R1	52	53	55	56
j loop	60	61			SW R5, 256(R0)	56	57	62	
lw r4,256(r0)	64	65	67	68	J I1	60	61		
lw r5,260(r0)	68	69	74	75	LW R4, 256(R0)	64	65	67	68
mul.d f0,f2,f4	72	73	79	80	LW R5, 260(R0)	68	69	74	75
beq r1,r2,done	76	77			MUL.D F0, F2, F4	72	73	79	80
dadd r1,r1,r1	80				BEQ R1, R2, I8	76	77		
div.d f10,f0,f6	84	85	91	92	DADD R1, R1, R1	80			
l.d f6, 1024(r2)	88	89	94	95	DIV.D F10, F0, F6	84	85	91	92
mul.d f20, f22, f24	92	93	99	100	LD F6, 1024(R2)	88	89	94	95
add.d f21, f24, f22	96	97	103	104	MUL.D F20, F22, F24	92	93	99	100
div.d f22, f28, f22	100	101	107	108	ADD.D F21, F24, F22	96	97	103	104
					DIV.D F22, F28, F22	100	101	107	108
Total number of access requests for instruction cache: 25					Total number of access requests for instruction cache: 25				
Number of instruction cache hits:0					Number of instruction cache hits: 0				
Total number of access requests for data cache: 9					Total number of access requests for data cache: 9				
Number of data cache hits:2					Number of data cache hits: 2				
					. saved results to file result.txt...				

```

Disassembly      1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
LW R4, 256(R0)           IF ID EX M1 M2 M3 M4 WB
LW R5, 260(R0)           IF ID EX M1 M2 M3 M4 WB
MUL.D F0, F2, F4        IF ID EX1 EX2 EX3 EX4 EX5 EX6 WB
BEQ R1, R2, I8           IF ID
DADD R1, R1, R1          IF ID EX M WB
SW R5, 256(R0)           IF ID EX M1 M2 M3 M4
J I1                     IF ID
DIV.D F10, F0, F6              IF
LW R4, 256(R0)
LW R5, 260(R0)
MUL.D F0, F2, F4
BEQ R1, R2, I8
DADD R1, R1, R1
SW R5, 256(R0)
J I1
LW R4, 256(R0)
LW R5, 260(R0)
MUL.D F0, F2, F4
BEQ R1, R2, I8
DADD R1, R1, R1
DIV.D F10, F0, F6
LD F6, 1024(R2)
MUL.D F20, F22, F24
ADD.D F21, F24, F22
DIV.D F22, F28, F22

Errors:
Exception: Divide by zero for instruction: DIV.D F10, F0, F6 !
Exception: Divide by zero for instruction: DIV.D F22, F28, F22 !

```

Only things that are additionally output by the program are two **exceptions** that are caused by **divide-by-zero** (since I assumed all FP registers have initial values zeros, since they are not loaded from any file, unlike the integer registers). Also, I don't immediately stop the programs upon receiving the exception (otherwise in this case the output would not have been complete), only I don't execute the operation that would have raised the exception (e.g., divide by zero in this case). Also, for this test case, I needed to change my parser a little bit, since I wrote the bison parser to support only the backward branch / jump, now I added forward jump too, that requires feeling the unresolved symbols either by an 1-pass assembly (I did this) or by maintaining a list or by a 2-pass assembly.

4. [Test case 2 modified \(CMSC611 Test Cases/Test 2 modified/inst.txt\)](#)

```

A: L.D F16, 1024(R4)
MUL.D F6,F7,F16
B: SUB.D F9,F16,F8
SUB.D F16,F1,F6
ADD.D F6,F2,F9
MUL.D F8,F9,F6
DADD R1,R5,R1
BEQ R1,R6,B
DADD R1,R5,R1
DSUB R2,R3,R2
BEQ R2,R7,A
DSUB R31,R31,R31

```

In this test case as we can see there are a few differences with the expected output (result_v1.txt). The first

difference that we can see is in the 5th instruction. Here, the instruction is to be fetched at $15 + (5 + 1) + (5 + 1) = 27^{\text{th}}$ cycle according to result_v1.txt, but according to the logic implemented, it's fetched at $24 + (5 + 1) + (5 + 1) = 36^{\text{th}}$ cycle.

Since IF and ID are **strictly sequential**, if there is **stall** before ID, then IF must **stall** (otherwise the earlier instruction that is waiting at ID will be overwritten, since it will be waiting in the IF-ID latch only). Since the **FPADD** unit is **non-pipelined** according to the configuration, 5th SUB.D must wait in the ID stage (i.e., before the ID stage, in the IF-ID latch, for it can't wait in the ID-EX latch, because that means it has already left ID stage, that it can't do before the 4th instruction leaves the FPADD execution unit, which it does at 24th cycle). As soon as the 5th instruction completes the ID stage, the 6th instruction starts for the IF stage, it takes another 12 cycles for cache miss, resulting in 36th cycle for the IF to be complete for the 6th instruction. This difference in delay with the expected output is propagated to the next instructions that increased the overall latency from every instruction after this instruction. The following figures (program outputs) explains the situation.

Expected Output					Actual Output				
l.d f16, 1024(r4)	12	13	20	21	LD F16, 1024(R4)	12	13	20	21
mul.d f6, f7, f16	13	14	26	27	MUL.D F6, F7, F16	13	14	26	27
sub.d f9, f16, f8	14	15	24	25	SUB.D F9, F16, F8	14	15	24	25
sub.d f16, f1, f6	15	24	30	31	SUB.D F16, F1, F6	15	24	30	31
add.d f6, f2, f9	27	30	34	35	ADD.D F6, F2, F9	35	36	40	42
mul.d f8, f9, f6	30	31	40	41	MUL.D F8, F9, F6	36	37	46	47
dadd r1, r5, r1	31	32	35	36	DADD R1, R5, R1	37	38	40	41
beq r1, r6, b	32	34			BEQ R1, R6, I3	38	40		
dadd r1, r5, r1	44				DADD R1, R5, R1	51			
sub.d f9, f16, f8	45	46	50	51	SUB.D F9, F16, F8	52	53	57	58
sub.d f16, f1, f6	46	50	54	55	SUB.D F16, F1, F6	53	57	61	62
add.d f6, f2, f9	50	54	58	59	ADD.D F6, F2, F9	57	61	65	67
mul.d f8, f9, f6	54	55	64	65	MUL.D F8, F9, F6	61	62	71	72
dadd r1, r5, r1	55	56	59	60	DADD R1, R5, R1	62	63	65	66
beq r1, r6, b	56	58			BEQ R1, R6, I3	63	65		
sub.d f9, f16, f8	58				SUB.D F9, F16, F8	65			
dadd r1, r5, r1	59	60	62	63	DADD R1, R5, R1	66	67	69	70
dsub r2, r3, r2	60	61	63	64	DSUB R2, R3, R2	67	68	70	71
beq r2, r7, a	61	63			BEQ R2, R7, I1	68	70		
dsub r31,r31,r31	63				DSUB R31, R31, R31	70			
l.d f16, 1024(r4)	64	65	67	68	LD F16, 1024(R4)	71	72	74	75
mul.d f6, f7, f16	65	66	73	74	MUL.D F6, F7, F16	72	73	80	81
sub.d f9, f16, f8	66	67	71	72	SUB.D F9, F16, F8	73	74	78	79
sub.d f16, f1, f6	67	71	77	78	SUB.D F16, F1, F6	74	78	84	85
add.d f6, f2, f9	71	77	81	82	ADD.D F6, F2, F9	78	84	88	90
mul.d f8, f9, f6	77	78	87	88	MUL.D F8, F9, F6	84	85	94	95
dadd r1, r5, r1	78	79	82	83	DADD R1, R5, R1	85	86	88	89
beq r1, r6, b	79	81			BEQ R1, R6, I3	86	88		
dadd r1, r5, r1	81	82	84	85	DADD R1, R5, R1	88	89	91	92
dsub r2, r3, r2	82	83	85	86	DSUB R2, R3, R2	89	90	92	93
beq r2, r7, a	83	85			BEQ R2, R7, I1	90	92		
l.d f16, 1024(r4)	85				LD F16, 1024(R4)	92			
dsub r31,r31,r31	86	87	89	90	DSUB R31, R31, R31	93	94	96	97
Total number of access requests for instruction cache: 33					Total number of access requests for instruction cache: 33				
Number of instruction cache hits:30					Number of instruction cache hits: 30				
Total number of access requests for data cache: 2					Total number of access requests for data cache: 2				
Number of data cache hits:1					Number of data cache hits: 1				

[illegible]

5. Test case 2 Config1 (CMSC Test Cases/Test 2 Config1/inst.txt)

As can be seen there is only one difference, that is again for the one stall before ID (at cycle 30, as opposed to after), as in the previous cases, this causes propagation of one-cycle extra delay than the one in result_v1.txt.

Disassembly	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
LD F16, 1024(R4)							IF	ID	EX	M1	M2	M3	M4	M5	M6	WB																				
MUL.D F6, F7, F16								IF	ID	S	S	S	S	S	S	EX1	EX2	EX3	EX4	EX5	EX6	WB														
SUB.D F9, F16, F8														IF	ID	EX1	EX2	EX3	EX4	WB																
SUB.D F16, F1, F6															IF	ID	S	S	S	S	S	EX1	EX2	EX3	EX4	WB										
ADD.D F6, F2, F9																					IF	ID	EX1	EX2	EX3	EX4	WB									
MUL.D F8, F9, F6																						IF	ID	S	S	S	EX1	EX2	EX3	EX4	EX5	EX6	WB			
DADD R1, R5, R1																													IF	ID	EX	M	WB			
BEQ R1, R6, I3																														IF	S	ID				
DADD R1, R5, R1																																				IF

Expected Output					Actual Output				
l.d f16, 1024(r4)	6	7	14	15	LD F16, 1024(R4)	6	7	14	15
mul.d f6, f7, f16	7	8	20	21	MUL.D F6, F7, F16	7	8	20	21
sub.d f9, f16, f8	13	14	18	19	SUB.D F9, F16, F8	13	14	18	19
sub.d f16, f1, f6	14	15	24	25	SUB.D F16, F1, F6	14	15	24	25
add.d f6, f2, f9	20	21	25	26	ADD.D F6, F2, F9	20	21	25	26
mul.d f8, f9, f6	21	22	31	32	MUL.D F8, F9, F6	21	22	31	32
dadd r1, r5, r1	27	28	30	31	DADD R1, R5, R1	27	28	30	31
beq r1, r6, b	28	30			BEQ R1, R6, I3	28	30		
dadd r1, r5, r1	34				DADD R1, R5, R1	35			
sub.d f9, f16, f8	35	36	40	41	SUB.D F9, F16, F8	36	37	41	42
sub.d f16, f1, f6	36	37	41	42	SUB.D F16, F1, F6	37	38	42	43
add.d f6, f2, f9	37	38	44	45	ADD.D F6, F2, F9	38	39	45	46
mul.d f8, f9, f6	38	39	50	51	MUL.D F8, F9, F6	39	40	51	52
dadd r1, r5, r1	39	40	42	43	DADD R1, R5, R1	40	41	43	44
beq r1, r6, b	40	42			BEQ R1, R6, I3	41	43		
sub.d f9, f16, f8	42				SUB.D F9, F16, F8	43			
dadd r1, r5, r1	43	44	46	47	DADD R1, R5, R1	44	45	47	48
dsub r2, r3, r2	44	45	47	48	DSUB R2, R3, R2	45	46	48	49
beq r2, r7, a	50	51			BEQ R2, R7, I1	51	52		
dsub r31,r31,r31	51				DSUB R31, R31, R31	52			
l.d f16, 1024(r4)	52	53	55	56	LD F16, 1024(R4)	53	54	56	57
mul.d f6, f7, f16	53	54	61	62	MUL.D F6, F7, F16	54	55	62	63
sub.d f9, f16, f8	54	55	59	60	SUB.D F9, F16, F8	55	56	60	61
sub.d f16, f1, f6	55	56	65	66	SUB.D F16, F1, F6	56	57	66	67
add.d f6, f2, f9	56	62	67	68	ADD.D F6, F2, F9	57	63	67	69
mul.d f8, f9, f6	62	63	72	73	MUL.D F8, F9, F6	63	64	73	74
dadd r1, r5, r1	63	64	66	67	DADD R1, R5, R1	64	65	67	68
beq r1, r6, b	64	66			BEQ R1, R6, I3	65	67		
dadd r1, r5, r1	66	67	69	70	DADD R1, R5, R1	67	68	70	71
dsub r2, r3, r2	67	68	70	71	DSUB R2, R3, R2	68	69	71	72
beq r2, r7, a	68	70			BEQ R2, R7, I1	69	71		
l.d f16, 1024(r4)	70				LD F16, 1024(R4)	71			
dsub r31,r31,r31	71	72	74	75	DSUB R31, R31, R31	72	73	75	76
Total number of access requests for instruction cache: 33					Total number of access requests for instruction cache: 33				
Number of instruction cache hits:27					Number of instruction cache hits: 27				
Total number of access requests for data cache: 2					Total number of access requests for data cache: 2				
Number of data cache hits:1					Number of data cache hits: 1				

6. Test case 2 Config2 (CMSC Test Cases/Test 2 Config2/inst.txt)

As before, the difference is again for the stalls before ID (at cycle 46, according to my implementation, the next fetch can start at this cycle, not before that since ID is in stall), as in the previous cases, this causes propagation of extra delays than the one in result_v1.txt as shown in the following results.

Expected Output					Actual Output				
l.d f16, 1024(r4)	20	21	42	43	LD F16, 1024(R4)	20	21	42	43
mul.d f6, f7, f16	22	23	48	49	MUL.D F6, F7, F16	21	22	48	49
sub.d f9, f16, f8	24	25	46	47	SUB.D F9, F16, F8	22	23	46	47
sub.d f16, f1, f6	26	46	52	53	SUB.D F16, F1, F6	23	46	52	53
add.d f6, f2, f9	46	52	56	57	ADD.D F6, F2, F9	65	66	70	72
mul.d f8, f9, f6	52	53	62	63	MUL.D F8, F9, F6	66	67	76	77
dadd r1, r5, r1	54	55	57	58	DADD R1, R5, R1	67	68	70	71
beq r1, r6, b	56	57			BEQ R1, R6, I3	68	70		
dadd r1, r5, r1	76				DADD R1, R5, R1	89			
sub.d f9, f16, f8	78	79	83	84	SUB.D F9, F16, F8	90	91	95	96
sub.d f16, f1, f6	80	83	87	88	SUB.D F16, F1, F6	91	95	99	100
add.d f6, f2, f9	83	87	91	92	ADD.D F6, F2, F9	95	99	103	105
mul.d f8, f9, f6	87	88	97	98	MUL.D F8, F9, F6	99	100	109	110
dadd r1, r5, r1	89	90	92	93	DADD R1, R5, R1	100	101	103	104
beq r1, r6, b	91	92			BEQ R1, R6, I3	101	103		
sub.d f9, f16, f8	93				SUB.D F9, F16, F8	103			
dadd r1, r5, r1	95	96	98	99	DADD R1, R5, R1	104	105	107	108
dsub r2, r3, r2	97	98	100	101	DSUB R2, R3, R2	105	106	108	109
beq r2, r7, a	99	100			BEQ R2, R7, I1	106	108		
dsub r31,r31,r31	101				DSUB R31, R31, R31	108			
l.d f16, 1024(r4)	103	104	107	108	LD F16, 1024(R4)	109	110	112	113
mul.d f6, f7, f16	105	106	113	114	MUL.D F6, F7, F16	110	111	118	119
sub.d f9, f16, f8	107	108	112	113	SUB.D F9, F16, F8	111	112	116	117
sub.d f16, f1, f6	109	112	117	118	SUB.D F16, F1, F6	112	116	122	123
add.d f6, f2, f9	112	117	121	122	ADD.D F6, F2, F9	116	122	126	128
mul.d f8, f9, f6	117	118	127	128	MUL.D F8, F9, F6	122	123	132	133
dadd r1, r5, r1	119	120	122	123	DADD R1, R5, R1	123	124	126	127
beq r1, r6, b	121	122			BEQ R1, R6, I3	124	126		
dadd r1, r5, r1	123	124	126	127	DADD R1, R5, R1	126	127	129	130
dsub r2, r3, r2	125	126	128	129	DSUB R2, R3, R2	127	128	130	131
beq r2, r7, a	127	128			BEQ R2, R7, I1	128	130		
l.d f16, 1024(r4)	129				LD F16, 1024(R4)	130			
dsub r31,r31,r31	131	132	134	135	DSUB R31, R31, R31	131	132	134	135
Total number of access requests for instruction cache: 33					Total number of access requests for instruction cache: 33				
Number of instruction cache hits:30					Number of instruction cache hits: 30				
Total number of access requests for data cache: 2					Total number of access requests for data cache: 2				
Number of data cache hits:1					Number of data cache hits: 1				

Disassembly	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
LD F16, 1024(R4)																				IF	ID	EX	S	S	S	S	S	S	S	S	S	S	S	S	S	M1
MUL.D F6, F7, F16																				IF	ID	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
SUB.D F9, F16, F8																					ID	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
SUB.D F16, F1, F6																					S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
ADD.D F6, F2, F9																					S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S

7. [Test cases 1/3/4 \(CMSC Test Cases/Test 1/3/4/inst.txt\)](#)

The outputs generated again differs a little bit from the ones generated in result_v1.txt. Some of them have some problem with termination. One can control the termination of the simulator by setting the **MAX_INS** macro, which is introduced to **prevent infinite loops**. Still the outputs that are generated are shown and compared to the expected output, they are nearly the same, as seen from the following output (for Test case 1). The one-cycle discrepancy is again due to the assumption of stall before ID.

Expected Output					Actual Output				
lw r0, 1024(r10)	5	6	13	14	LW R0, 1024(R10)	5	6	13	14
lw r1, 1024(r11)	10	11	14	15	LW R1, 1024(R11)	10	11	14	15
lw r2, 1024(r12)	15	16	23	24	LW R2, 1024(R12)	15	16	23	24
lw r4, 1024(r13)	20	21	24	25	LW R4, 1024(R13)	20	21	24	25
dsub r0, r0, r1	25	26	28	29	DSUB R0, R0, R1	25	26	28	29
add.d f2, f4, f6	30	31	35	36	ADD.D F2, F4, F6	30	31	35	36
bne r0, r2, l1	35	36			BNE R0, R2, I5	35	36		
sw r0, 1024(r14)	40				SW R0, 1024(R14)	40			
dsub r0, r0, r1	41	42	44	45	DSUB R0, R0, R1	41	42	44	45
add.d f2, f4, f6	42	43	48	49	ADD.D F2, F4, F6	42	43	47	49
bne r0, r2, l1	43	44			BNE R0, R2, I5	43	44		
dsub r0, r0, r1	44	45	47	48	DSUB R0, R0, R1	44	45	47	48
add.d f2, f4, f6	45	48	53	54	ADD.D F2, F4, F6	45	47	51	53
bne r0, r2, l1	48	49			BNE R0, R2, I5	47	48		
dsub r0, r0, r1	49	50	52	53	DSUB R0, R0, R1	48	49	51	52
add.d f2, f4, f6	50	53	57	58	ADD.D F2, F4, F6	49	51	55	56
bne r0, r2, l1	53	54			BNE R0, R2, I5	51	52		
dsub r0, r0, r1	54				DSUB R0, R0, R1	52			
sw r0, 1024(r14)	55	56	63		SW R0, 1024(R14)	53	54	61	
lw r3, 1024(r9)	60	61	64	65	LW R3, 1024(R9)	58	59	62	63
dsub r0, r0, r1	65	66	68	69	DSUB R0, R0, R1	63	64	66	67
dsub r2, r2, r1	70	71	73	74	DSUB R2, R2, R1	68	69	71	72
mul.d f8, f2, f4	75	76	82	83	MUL.D F8, F2, F4	73	74	80	81
add.d f8, f8, f10	80	82	86	87	ADD.D F8, F8, F10	78	80	84	85
dadd r14,r14,r4	85	86	88	89	DADD R14, R14, R4	84	85	87	88
bne r0, r3, l2	90	91			BNE R0, R3, I6	89	90		
dadd r0, r1, r3	95				DADD R0, R1, R3	94			
add.d f2, f4, f6	100	101	105	106	ADD.D F2, F4, F6	99	100	104	105
bne r0, r2, l1	105	106			BNE R0, R2, I5	104	105		
sw r0, 1024(r14)	110	111	118		SW R0, 1024(R14)	109	110	117	
lw r3, 1024(r9)	115	116	119	120	LW R3, 1024(R9)	114	115	118	119
dsub r0, r0, r1	120	121	123	124	DSUB R0, R0, R1	119	120	122	123
dsub r2, r2, r1	125	126	128	129	DSUB R2, R2, R1	124	125	127	128
mul.d f8, f2, f4	130	131	137	138	MUL.D F8, F2, F4	129	130	136	137
add.d f8, f8, f10	135	137	141	142	ADD.D F8, F8, F10	134	136	140	141
dadd r14,r14,r4	140	141	143	144	DADD R14, R14, R4	140	141	143	144
bne r0, r3, l2	145	146			BNE R0, R3, I6	145	146		
add.d f2, f4, f6	150				ADD.D F2, F4, F6	150	151	155	156
dadd r0, r1, r3	155	156	158	159	BNE R0, R2, I5	155	156		

The following figures present the output fragments for Test_4 (the output consists of a huge number of lines, first few lines are shown here, again the **maximum number of instructions** to be executed can be controlled by the **MAX_INS** macro in cpu.h), which the simulator generates the following exception as well: "Exception: out of range access with address: 290 (should be in [256, 287]), Instruction: SD F3, 8(R29)!"

[illegible]

Expected Output					Actual Output				
lw r1, 0(r31)	9	10	18	19	LW R1, 0(R31)	9	10	18	19
lw r4, 20(r31)	11	12	25	26	LW R4, 20(R31)	10	11	25	26
lw r3, 28(r31)	13	18	32	33	LW R3, 28(R31)	11	12	32	33
dsub r2,r4,r3	18	25	34	35	DSUB R2, R4, R3	12	13	34	35
l.d f2, 4(r31)	27	33	36	37	LD F2, 4(R31)	21	33	35	36
l.d f3, 12(r31)	33	34	43	44	LD F3, 12(R31)	33	34	42	43
l.d f4, 12(r31)	35	36	45	46	LD F4, 12(R31)	34	35	43	44
add.d f3,f3,f2	37	43	47	48	ADD.D F3, F3, F2	35	42	46	47
mul.d f4,f4,f2	46	47	53	54	MUL.D F4, F4, F2	50	51	57	58
l.d f5, 12(r31)	48	49	52	53	LD F5, 12(R31)	51	52	54	55
div.d f5,f5,f2	50	52	62	63	DIV.D F5, F5, F2	52	54	64	65
s.d f3, -8(r29)	52	53	61		SD F3, 8(R29)	54	55	57	
s.d f4, -8(r29)	61	62	65		SD F4, 8(R29)	63	64	66	
s.d f5, -8(r29)	63	64	67		SD F5, 8(R29)	64	65	67	
dsub r1,r1,r2	65	66	68	69	DSUB R1, R1, R2	65	66	68	69
bne r0,r1,loop	67	68			BNE R0, R1, I5	66	68		
j end	76				J I0	76			
l.d f2, 4(r31)	78	79	82	83	LD F2, 4(R31)	77	78	80	81
l.d f3, 12(r31)	80	81	84	85	LD F3, 12(R31)	78	79	81	82
l.d f4, 12(r31)	82	83	86	87	LD F4, 12(R31)	79	80	82	83
add.d f3,f3,f2	84	85	89	90	ADD.D F3, F3, F2	80	81	85	87
mul.d f4,f4,f2	86	87	93	94	MUL.D F4, F4, F2	81	82	88	89
l.d f5, 12(r31)	88	89	92	93	LD F5, 12(R31)	82	83	85	86
div.d f5,f5,f2	90	92	102	103	DIV.D F5, F5, F2	83	85	95	96
s.d f3, -8(r29)	92	93	96		SD F3, 8(R29)	85	86	88	
s.d f4, -8(r29)	94	95	98		SD F4, 8(R29)	86	87	89	
s.d f5, -8(r29)	96	97	104		SD F5, 8(R29)	87	88	96	
dsub r1,r1,r2	98	99	105	106	DSUB R1, R1, R2	88	89	97	98
bne r0,r1,loop	100	101			BNE R0, R1, I5	89	91		
l.d f2, 4(r31)	102	104	107	108	LD F2, 4(R31)	91	92	98	99
l.d f3, 12(r31)	104	105	109	110	LD F3, 12(R31)	92	93	99	100
l.d f4, 12(r31)	106	107	111	112	LD F4, 12(R31)	93	94	100	101
add.d f3,f3,f2	108	109	113	114	ADD.D F3, F3, F2	94	99	103	105
mul.d f4,f4,f2	110	111	117	118	MUL.D F4, F4, F2	99	100	106	107
l.d f5, 12(r31)	112	113	116	117	LD F5, 12(R31)	100	101	103	104
div.d f5,f5,f2	114	116	126	127	DIV.D F5, F5, F2	101	103	113	114
s.d f3, -8(r29)	116	117	120		SD F3, 8(R29)	103	104	106	

8. Test Case done in the class with configurations from the project document

```
A: L.D F16, 1032(R4)
MUL.D F6, F7, F16
B: SUB.D F9, F16, F8
SUB.D F16, F1, F6
ADD.D F6, F2, F4
MUL.D F8, F4, F6
DADD R1, R5, R1
BNE R1, R6, B
```

[illegible]

LD F16, 1032(R4)	6	7	14	15
MUL.D F6, F7, F16	7	8	20	21
SUB.D F9, F16, F8	13	14	18	19
SUB.D F16, F1, F6	14	15	24	25
ADD.D F6, F2, F4	20	21	25	26
MUL.D F8, F4, F6	21	22	31	32
DADD R1, R5, R1	27	28	30	31
BNE R1, R6, I3	28	30		
SUB.D F9, F16, F8	36	37	41	42
SUB.D F16, F1, F6	37	38	42	43
ADD.D F6, F2, F4	38	39	43	45
MUL.D F8, F4, F6	39	40	49	50
DADD R1, R5, R1	40	41	43	44
BNE R1, R6, I3	41	43		
SUB.D F9, F16, F8	43	44	53	54
SUB.D F16, F1, F6	44	50	54	55
ADD.D F6, F2, F4	50	51	55	57
MUL.D F8, F4, F6	51	52	61	62
DADD R1, R5, R1	52	53	55	56
BNE R1, R6, I3	53	55		
SUB.D F9, F16, F8	55	56	65	66
SUB.D F16, F1, F6	56	62	66	67
ADD.D F6, F2, F4	62	63	67	69
MUL.D F8, F4, F6	63	64	73	74
DADD R1, R5, R1	64	65	67	68
BNE R1, R6, I3	65	67		
SUB.D F9, F16, F8	67	68	77	78
SUB.D F16, F1, F6	68	74	78	79
ADD.D F6, F2, F4	74	75	79	81
MUL.D F8, F4, F6	75	76	85	86
DADD R1, R5, R1	76	77	79	80
BNE R1, R6, I3	77	79		

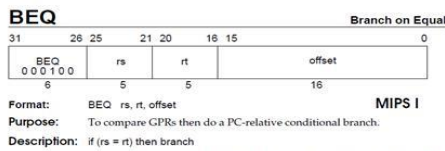
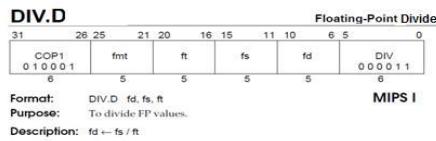
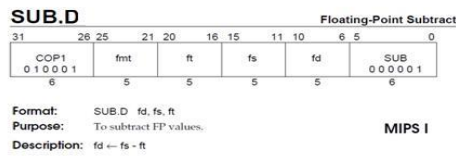
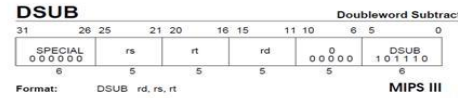
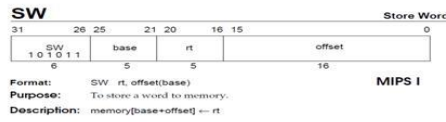
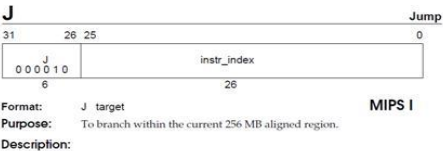
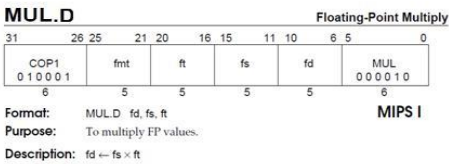
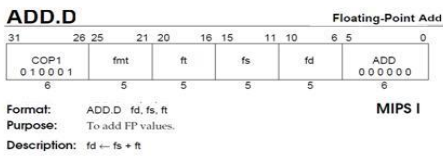
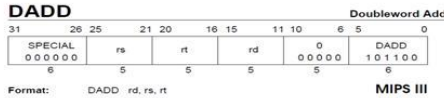
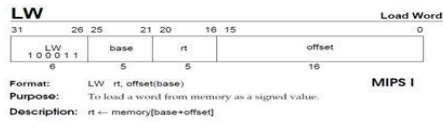
Couple of important things to be mentioned:

- 1) If there is a parse error, a syntax error message will be generated from the parser.
- 2) A HALT instruction is inserted silently at the end of all instruction (to detect the end of execution).
- 3) log.txt and pipeline.txt can be viewed to find useful information about step by step execution of instructions / hazards / cache contents / BTB contents / registers / memory and everything.
- 4) result of the execution in desired format is written in result.txt.

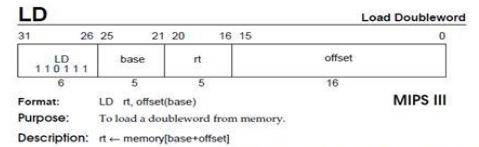
The supported subset of MIPS instruction set

Instruction Class	Instruction Mnemonic
Data Transfers	LW, SW, LD, SD
Arithmetic/ logical	DADD, DSUB, ADD.D, MUL.D, DIV.D, SUB.D
Control	J, BEQ, BNE

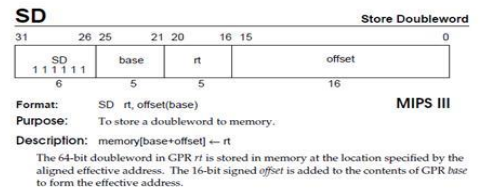
Reduced MIPS instruction set



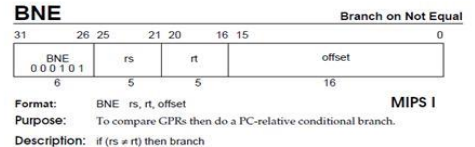
An 18-bit signed offset (the 16-bit *offset* field shifted left 2 bits) is added to the address of the instruction following the branch (not the branch itself), in the branch delay slot, to form a PC-relative effective target address.
 If the contents of GPR *rs* and GPR *rt* are equal, branch to the effective target address after the instruction in the delay slot is executed.



The contents of the 64-bit doubleword at the memory location specified by the aligned effective address are fetched and placed in GPR *rt*. The 16-bit signed *offset* is added to the contents of GPR *base* to form the effective address.



The 64-bit doubleword in GPR *rt* is stored in memory at the location specified by the aligned effective address. The 16-bit signed *offset* is added to the contents of GPR *base* to form the effective address.



An 18-bit signed offset (the 16-bit *offset* field shifted left 2 bits) is added to the address of the instruction following the branch (not the branch itself), in the branch delay slot, to form a PC-relative effective target address.
 If the contents of GPR *rs* and GPR *rt* are not equal, branch to the effective target address after the instruction in the delay slot is executed.

The grammar for MIPS instruction set assembler

Instruction_List \rightarrow *Instruction_List* *Instruction* | ϵ
Instruction \rightarrow *DataTransferInstruction* | *ArithmeticLogicalInstruction* | *ControlInstruction*
DataTransferInstruction \rightarrow *DTOperation* *DTOperands*
ArithmeticLogicalInstruction \rightarrow *ALUOperation* *ALUOperands*
Control \rightarrow *Jump* | *Branch*
Jump \rightarrow J *Address*
Branch \rightarrow *BranchOperation* *BranchOperands*
DTOperation \rightarrow "LW" | "LD" | "SW" | "SD"
DTOperands \rightarrow REG '*i*' *Address* '(' GPR '*j*')
ALUOperation \rightarrow "DADD" | "DSUB" | "ADD.D" | "SUB.D" | "MULT.D" | "DIV.D"
ALUOperands \rightarrow REG '*i*' , REG '*j*' , REG '*k*'
BranchOperation \rightarrow "BEQ" | "BNE"
BranchOperands \rightarrow REG '*i*' , REG '*j*' , *Address*
Address \rightarrow [0-9]+
REG \rightarrow GPR | FPR
GPR \rightarrow R[0-9]+
FPR \rightarrow F[0-9]+

The GUI Version of the simulator

[illegible]