CSE 3038 COMPUTER ORGANIZATION

PROJECT 2

REPORT

BORA DUMAN 150121043

FURKAN GÖKGÖZ 150120076

ARDA ÖZTÜRK 150121051

1. Introduction and Objective

This report provides a detailed overview of the enhancements made to the MIPS processor architecture with the given new instructions.

2. New Instructions' Implementation Details

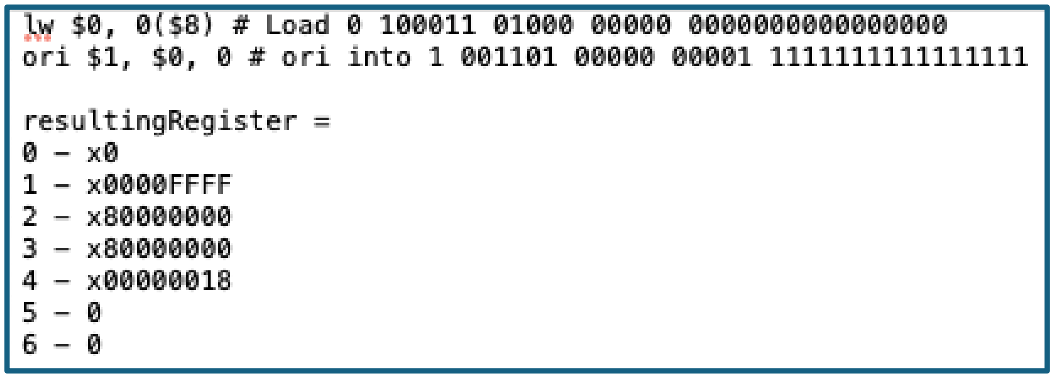
The six instructions that have been given will be talked in detail below. Custom test cases were made for each, and their expected results were decided by the certain way they were implemented and the way they have been called in instructions.

2.1 Ori

Ori is an I-type instruction where the immediate value is zero-extended and then OR-ed with the contents of a source register. The result is stored in the destination register.

For this, we have added a zero-extender under the sign-extend component in the datapath, connected to imm[15-0]. The result of ALU and this imm-value was OR-ed and put in relevant register with new control signal "ori" triggering RegWrite, ALUSrc and ALUOp1 control signals.

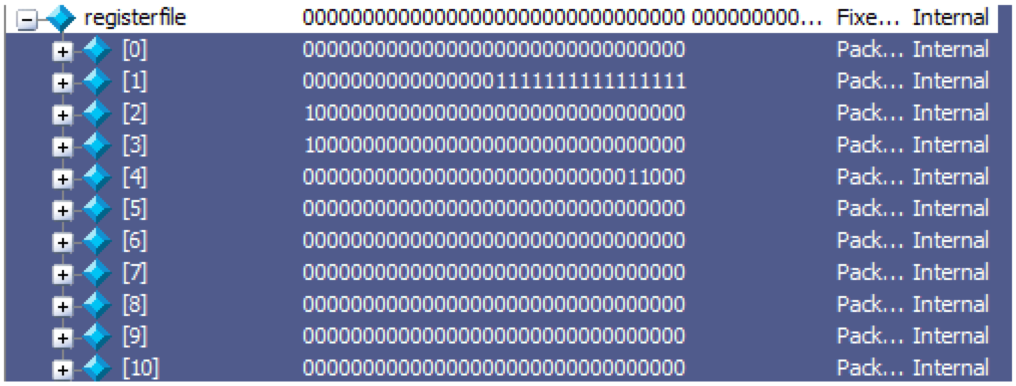
2.1.1 Test Case and Expected Result



```
lw $0, 0($8) # Load 0 100011 01000 00000 0000000000000000
ori $1, $0, 0 # ori into 1 001101 00000 00001 1111111111111111

resultingRegister =
0 - x0
1 - x0000FFFF
2 - x80000000
3 - x80000000
4 - x00000018
5 - 0
6 - 0
```

Figure 1. Ori test case and results in register.

2.1.2 Register Result



```
registerfile  00000000000000000000000000000000 000000000...  Fixe...  Internal
  [0]         00000000000000000000000000000000               Pack...  Internal
  [1]         00000000000000001111111111111111               Pack...  Internal
  [2]         10000000000000000000000000000000               Pack...  Internal
  [3]         10000000000000000000000000000000               Pack...  Internal
  [4]         00000000000000000000000000011000               Pack...  Internal
  [5]         00000000000000000000000000000000               Pack...  Internal
  [6]         00000000000000000000000000000000               Pack...  Internal
  [7]         00000000000000000000000000000000               Pack...  Internal
  [8]         00000000000000000000000000000000               Pack...  Internal
  [9]         00000000000000000000000000000000               Pack...  Internal
  [10]        00000000000000000000000000000000               Pack...  Internal
```

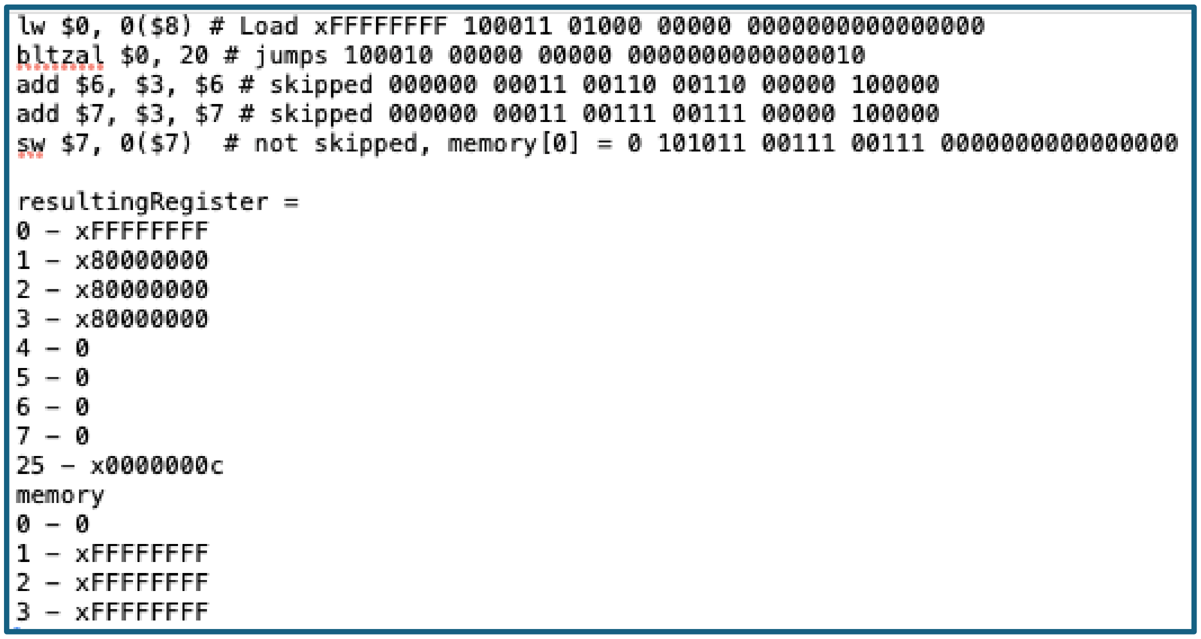
Figure 2. The register file after Ori.

2.2 Bltzal

Bltzal is an I-type instruction that evaluates the sign of the content in the source register. If negative, the processor branches to the address computed from the current PC and signed offset, while the return address is saved in register $25.

For this, we have added a new negative signal into ALU, which it outputs. Along with a new "bltzal" control signal, we check the negativity and if both are true (with an AND gate) and if branching signals are true the operation gets triggered, with RegWrite control signal also having part in linking.

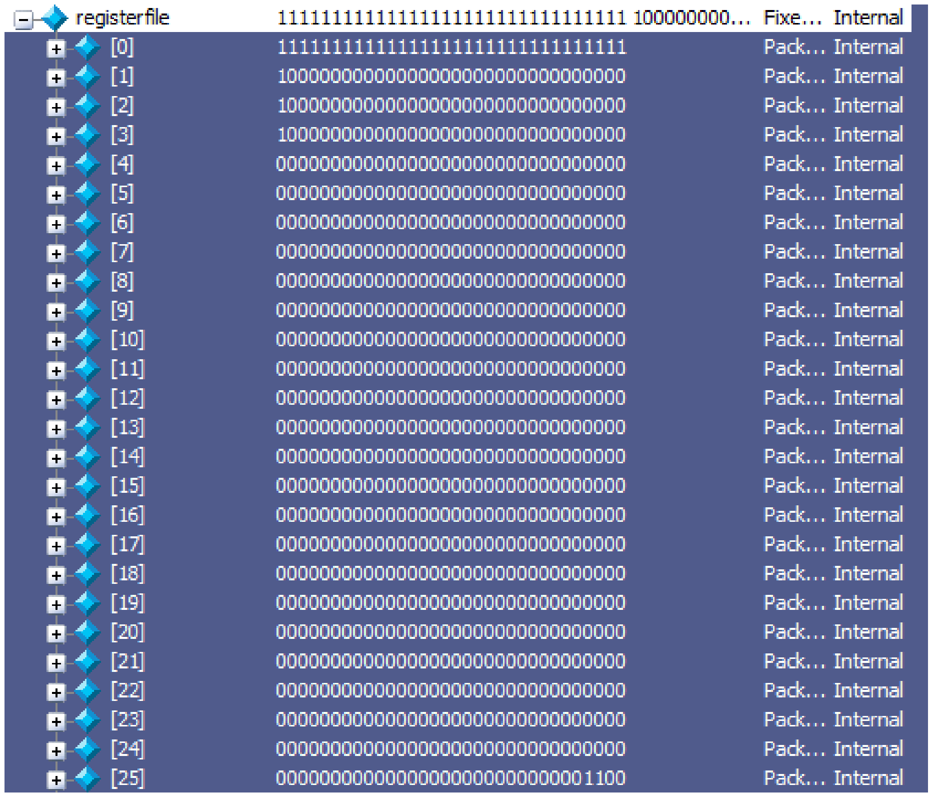
2.2.1 Test Case and Expected Result



```
lw $0, 0($8) # Load xFFFFFFFF 100011 01000 00000 0000000000000000
bltzal $0, 20 # jumps 100010 00000 00000 0000000000000010
add $6, $3, $6 # skipped 000000 00011 00110 00110 00000 100000
add $7, $3, $7 # skipped 000000 00011 00111 00111 00000 100000
sw $7, 0($7)  # not skipped, memory[0] = 0 101011 00111 00111 0000000000000000

resultingRegister =
0 - xFFFFFFFF
1 - x80000000
2 - x80000000
3 - x80000000
4 - 0
5 - 0
6 - 0
7 - 0
25 - x0000000c
memory
0 - 0
1 - xFFFFFFFF
2 - xFFFFFFFF
3 - xFFFFFFFF
```

Figure 3. Bltzal test case, and results in register and memory.

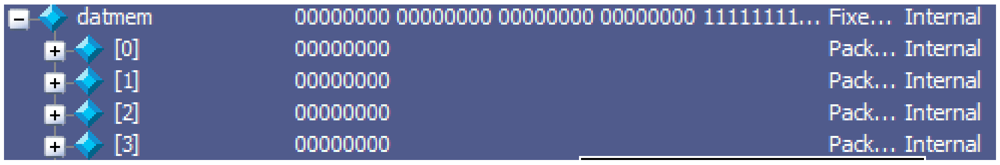
2.2.2 Register Result



```
registerfile  11111111111111111111111111111111 100000000...  Fixe...  Internal
  [0]         11111111111111111111111111111111               Pack...  Internal
  [1]         10000000000000000000000000000000               Pack...  Internal
  [2]         10000000000000000000000000000000               Pack...  Internal
  [3]         10000000000000000000000000000000               Pack...  Internal
  [4]         00000000000000000000000000000000               Pack...  Internal
  [5]         00000000000000000000000000000000               Pack...  Internal
  [6]         00000000000000000000000000000000               Pack...  Internal
  [7]         00000000000000000000000000000000               Pack...  Internal
  [8]         00000000000000000000000000000000               Pack...  Internal
  [9]         00000000000000000000000000000000               Pack...  Internal
  [10]        00000000000000000000000000000000               Pack...  Internal
  [11]        00000000000000000000000000000000               Pack...  Internal
  [12]        00000000000000000000000000000000               Pack...  Internal
  [13]        00000000000000000000000000000000               Pack...  Internal
  [14]        00000000000000000000000000000000               Pack...  Internal
  [15]        00000000000000000000000000000000               Pack...  Internal
  [16]        00000000000000000000000000000000               Pack...  Internal
  [17]        00000000000000000000000000000000               Pack...  Internal
  [18]        00000000000000000000000000000000               Pack...  Internal
  [19]        00000000000000000000000000000000               Pack...  Internal
  [20]        00000000000000000000000000000000               Pack...  Internal
  [21]        00000000000000000000000000000000               Pack...  Internal
  [22]        00000000000000000000000000000000               Pack...  Internal
  [23]        00000000000000000000000000000000               Pack...  Internal
  [24]        00000000000000000000000000000000               Pack...  Internal
  [25]        00000000000000000000000000001100               Pack...  Internal
```

Figure 4. The register file after Bltzal.

2.2.3 Memory Result



```
datmem   00000000 00000000 00000000 00000000 11111111...  Fixe...  Internal
  [0]    00000000                                         Pack...  Internal
  [1]    00000000                                         Pack...  Internal
  [2]    00000000                                         Pack...  Internal
  [3]    00000000                                         Pack...  Internal
```

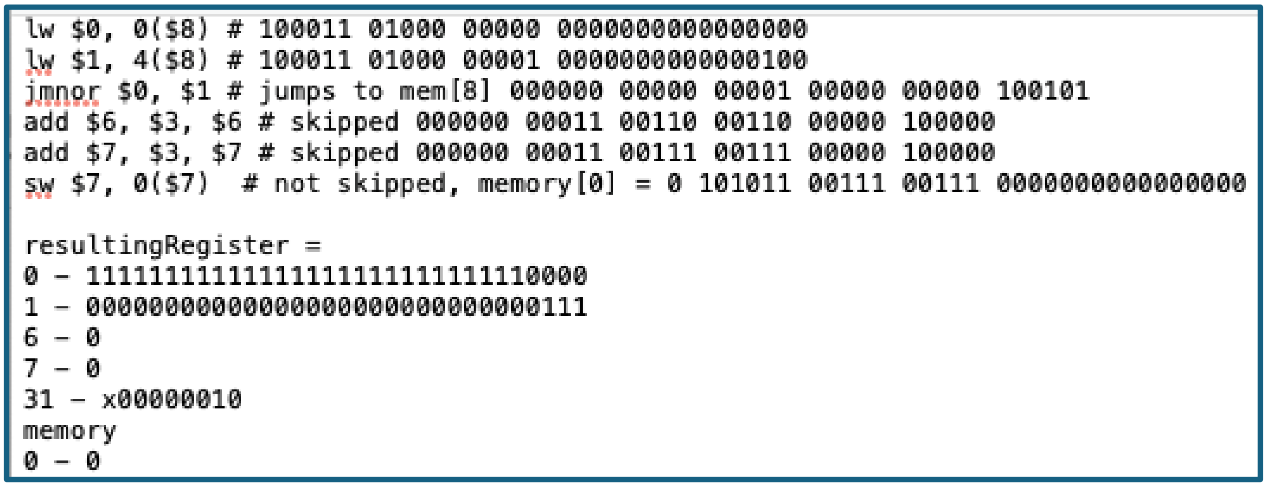
Figure 5. The data memory after Bltzal.

2.3 Jmnor

Jmnor is an R-type instruction that uses the NOR operation on two specified registers. The result is used to update the program counter and to store in register $31.

For this, we have updated the ALU for it to have a NOR operation. A “jmnor” wire is added in the processor for signaling. The RegWrite signal is triggered for link addressing.

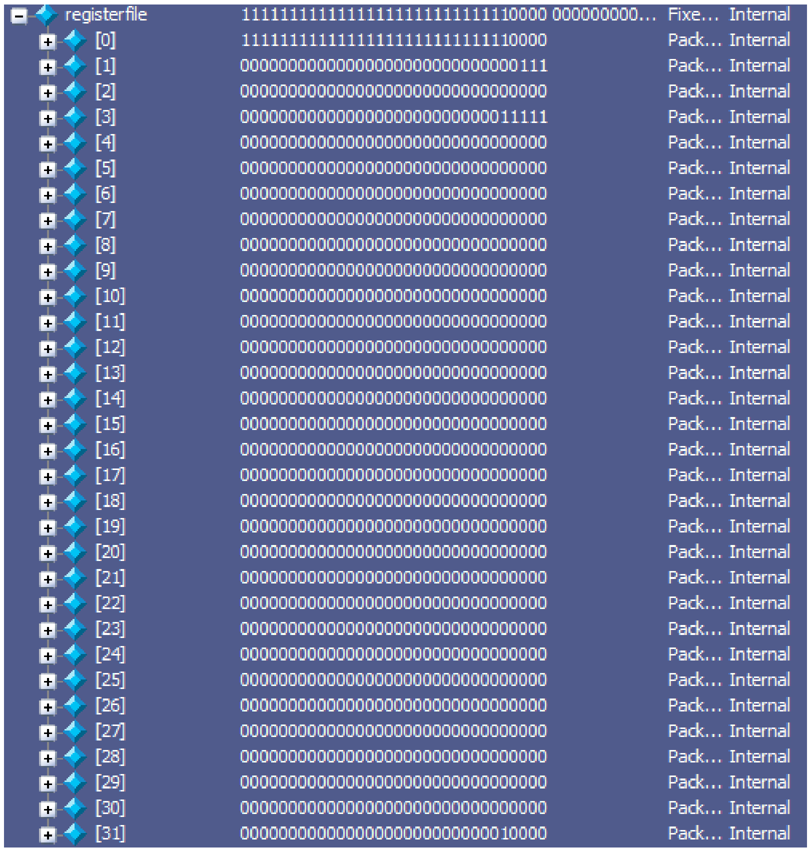
2.3.1 Test Case and Expected Result



```
lw $0, 0($8) # 100011 01000 00000 0000000000000000
lw $1, 4($8) # 100011 01000 00001 0000000000000100
jmnor $0, $1 # jumps to mem[8] 000000 00000 00001 00000 00000 100101
add $6, $3, $6 # skipped 000000 00011 00110 00110 00000 100000
add $7, $3, $7 # skipped 000000 00011 00111 00111 00000 100000
sw $7, 0($7)  # not skipped, memory[0] = 0 101011 00111 00111 0000000000000000

resultingRegister =
0 - 11111111111111111111111111110000
1 - 00000000000000000000000000000111
6 - 0
7 - 0
31 - x00000010
memory
0 - 0
```

Figure 6. Jmnor test case, and results in register and memory.

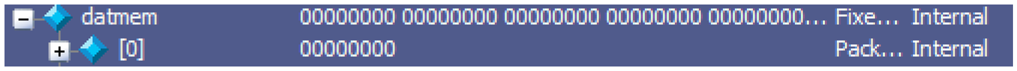
2.3.2 Register Result



```
registerfile  11111111111111111111111111110000 000000000...  Fixe... Internal
  [0]         11111111111111111111111111110000               Pack... Internal
  [1]         00000000000000000000000000000111               Pack... Internal
  [2]         00000000000000000000000000000000               Pack... Internal
  [3]         00000000000000000000000000011111               Pack... Internal
  [4]         00000000000000000000000000000000               Pack... Internal
  [5]         00000000000000000000000000000000               Pack... Internal
  [6]         00000000000000000000000000000000               Pack... Internal
  [7]         00000000000000000000000000000000               Pack... Internal
  [8]         00000000000000000000000000000000               Pack... Internal
  [9]         00000000000000000000000000000000               Pack... Internal
  [10]        00000000000000000000000000000000               Pack... Internal
  [11]        00000000000000000000000000000000               Pack... Internal
  [12]        00000000000000000000000000000000               Pack... Internal
  [13]        00000000000000000000000000000000               Pack... Internal
  [14]        00000000000000000000000000000000               Pack... Internal
  [15]        00000000000000000000000000000000               Pack... Internal
  [16]        00000000000000000000000000000000               Pack... Internal
  [17]        00000000000000000000000000000000               Pack... Internal
  [18]        00000000000000000000000000000000               Pack... Internal
  [19]        00000000000000000000000000000000               Pack... Internal
  [20]        00000000000000000000000000000000               Pack... Internal
  [21]        00000000000000000000000000000000               Pack... Internal
  [22]        00000000000000000000000000000000               Pack... Internal
  [23]        00000000000000000000000000000000               Pack... Internal
  [24]        00000000000000000000000000000000               Pack... Internal
  [25]        00000000000000000000000000000000               Pack... Internal
  [26]        00000000000000000000000000000000               Pack... Internal
  [27]        00000000000000000000000000000000               Pack... Internal
  [28]        00000000000000000000000000000000               Pack... Internal
  [29]        00000000000000000000000000000000               Pack... Internal
  [30]        00000000000000000000000000000000               Pack... Internal
  [31]        00000000000000000000000000010000               Pack... Internal
```

Figure 7. The register file after Jmnor.

2.3.3 Memory Result



```
datmem      00000000 00000000 00000000 00000000 00000000... Fixe... Internal
  [0]       00000000                                        Pack... Internal
```

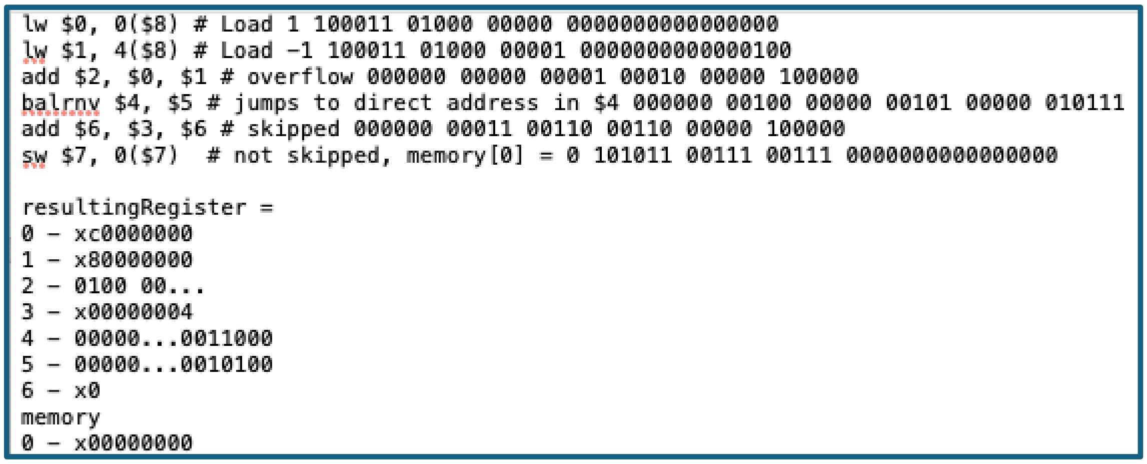
Figure 8. The data memory after Jmnor.

2.4 Balrnv

Balrnv is an R-type instruction that checks the V-status signal, and if zero, branches to address in $rs.

For this, we have added a new V-status flag and its checks in the processor file. A "balrnv" wire was added into the processor for signaling.

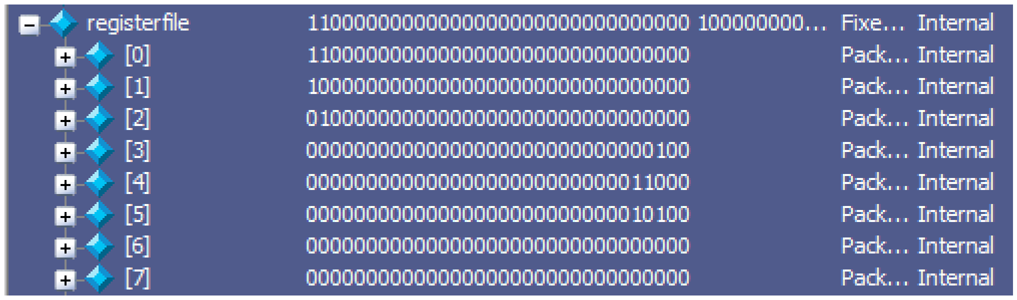
2.4.1 Test Case and Expected Result



```
lw $0, 0($8) # Load 1 100011 01000 00000 0000000000000000
lw $1, 4($8) # Load -1 100011 01000 00001 0000000000000100
add $2, $0, $1 # overflow 000000 00000 00001 00010 00000 100000
balrnv $4, $5 # jumps to direct address in $4 000000 00100 00000 00101 00000 010111
add $6, $3, $6 # skipped 000000 00011 00110 00110 00000 100000
sw $7, 0($7)  # not skipped, memory[0] = 0 101011 00111 00111 0000000000000000

resultingRegister =
0 - xc0000000
1 - x80000000
2 - 0100 00...
3 - x00000004
4 - 00000...0011000
5 - 00000...0010100
6 - x0
memory
0 - x00000000
```

Figure 9. Balrnv test case, and results in register and memory.

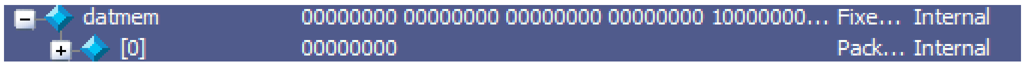
2.4.2 Register Result



```
registerfile   11000000000000000000000000000000 100000000... Fixe... Internal
  [0]          11000000000000000000000000000000              Pack... Internal
  [1]          10000000000000000000000000000000              Pack... Internal
  [2]          01000000000000000000000000000000              Pack... Internal
  [3]          00000000000000000000000000000100              Pack... Internal
  [4]          00000000000000000000000000011000              Pack... Internal
  [5]          00000000000000000000000000010100              Pack... Internal
  [6]          00000000000000000000000000000000              Pack... Internal
  [7]          00000000000000000000000000000000              Pack... Internal
```

Figure 10. The register file after Balrnv.

2.4.3 Memory Result



```
datmem    00000000 00000000 00000000 00000000 10000000... Fixe... Internal
  [0]     00000000                                        Pack... Internal
```

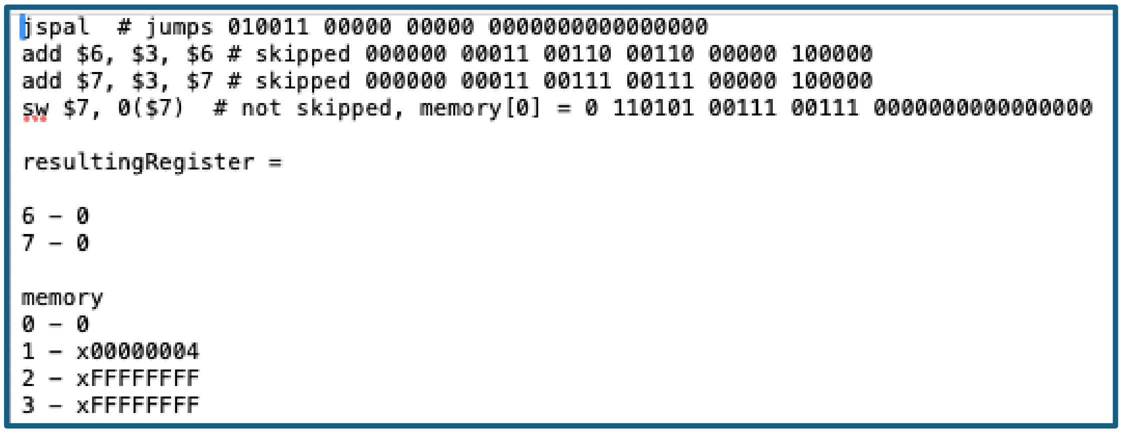
Figure 11. The data memory after Balrnv.

2.5 JSPAL

JSPAL is an I-type instruction that jumps to address written in $sp and stores link address in memory.

For this, we trigger the MemWrite. We have also added a new “jspal” control signal.

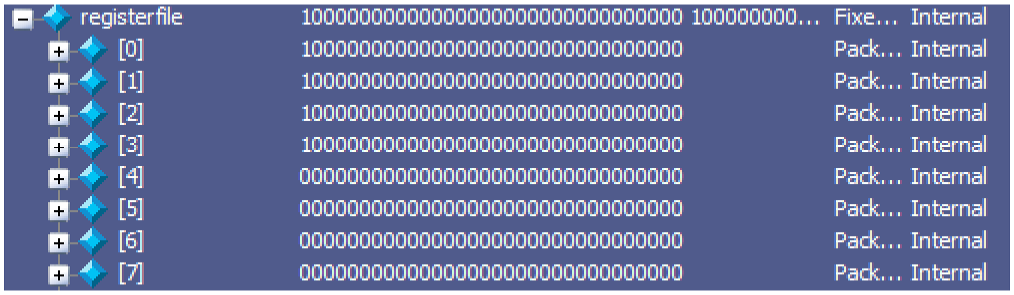
2.5.1 Test Case and Expected Result



```
jspal  # jumps 010011 00000 00000 0000000000000000
add $6, $3, $6 # skipped 000000 00011 00110 00110 00000 100000
add $7, $3, $7 # skipped 000000 00011 00111 00111 00000 100000
sw $7, 0($7)  # not skipped, memory[0] = 0 110101 00111 00111 0000000000000000

resultingRegister =

6 - 0
7 - 0

memory
0 - 0
1 - x00000004
2 - xFFFFFFFF
3 - xFFFFFFFF
```

Figure 12. Jspal test case, and results in register and memory.

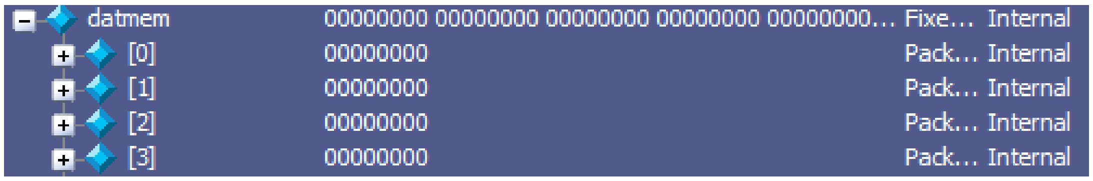
2.5.2 Register Result



```
registerfile  10000000000000000000000000000000 100000000... Fixe... Internal
  [0]         10000000000000000000000000000000              Pack... Internal
  [1]         10000000000000000000000000000000              Pack... Internal
  [2]         10000000000000000000000000000000              Pack... Internal
  [3]         10000000000000000000000000000000              Pack... Internal
  [4]         00000000000000000000000000000000              Pack... Internal
  [5]         00000000000000000000000000000000              Pack... Internal
  [6]         00000000000000000000000000000000              Pack... Internal
  [7]         00000000000000000000000000000000              Pack... Internal
```

Figure 13. The register file after Jspal.

2.5.3 Memory Result



```
datmem   00000000 00000000 00000000 00000000 00000000... Fixe... Internal
  [0]    00000000                                       Pack... Internal
  [1]    00000000                                       Pack... Internal
  [2]    00000000                                       Pack... Internal
  [3]    00000000                                       Pack... Internal
```

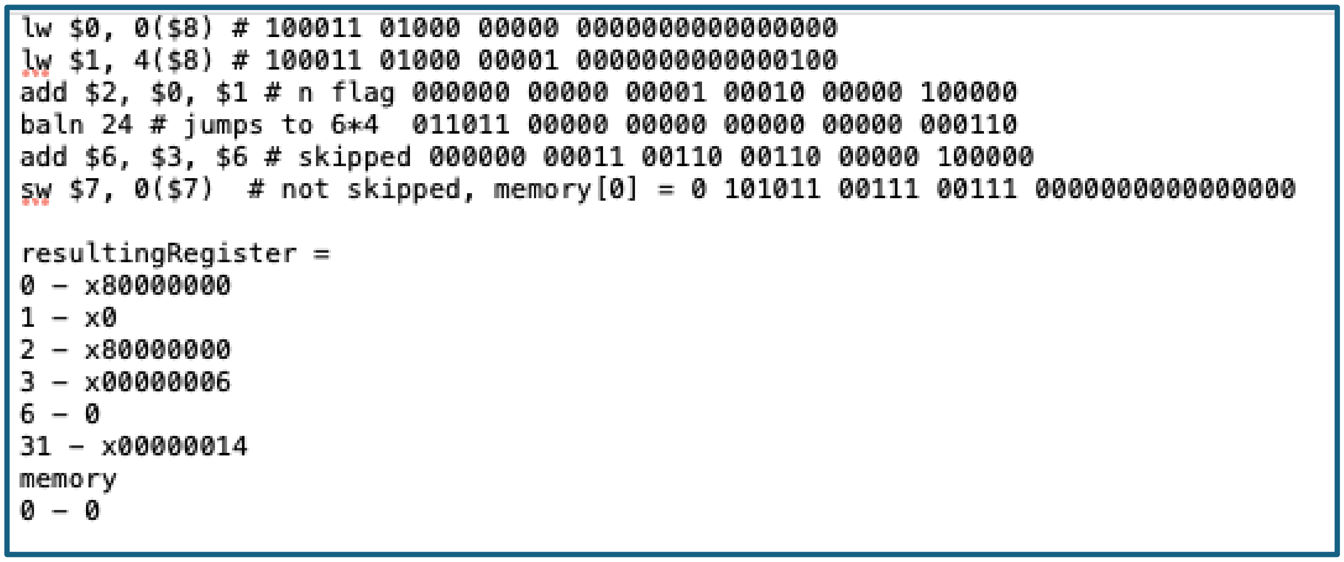
Figure 14. The data memory after Jspal.

2.6 BALN

BALN is a J-type instruction that checks N-status signal, and if one, branches to pseudo-direct address and stores link address in register.

For this, we trigger the RegWrite. The N-status flag has been integrated in the processor file alongside its checks.

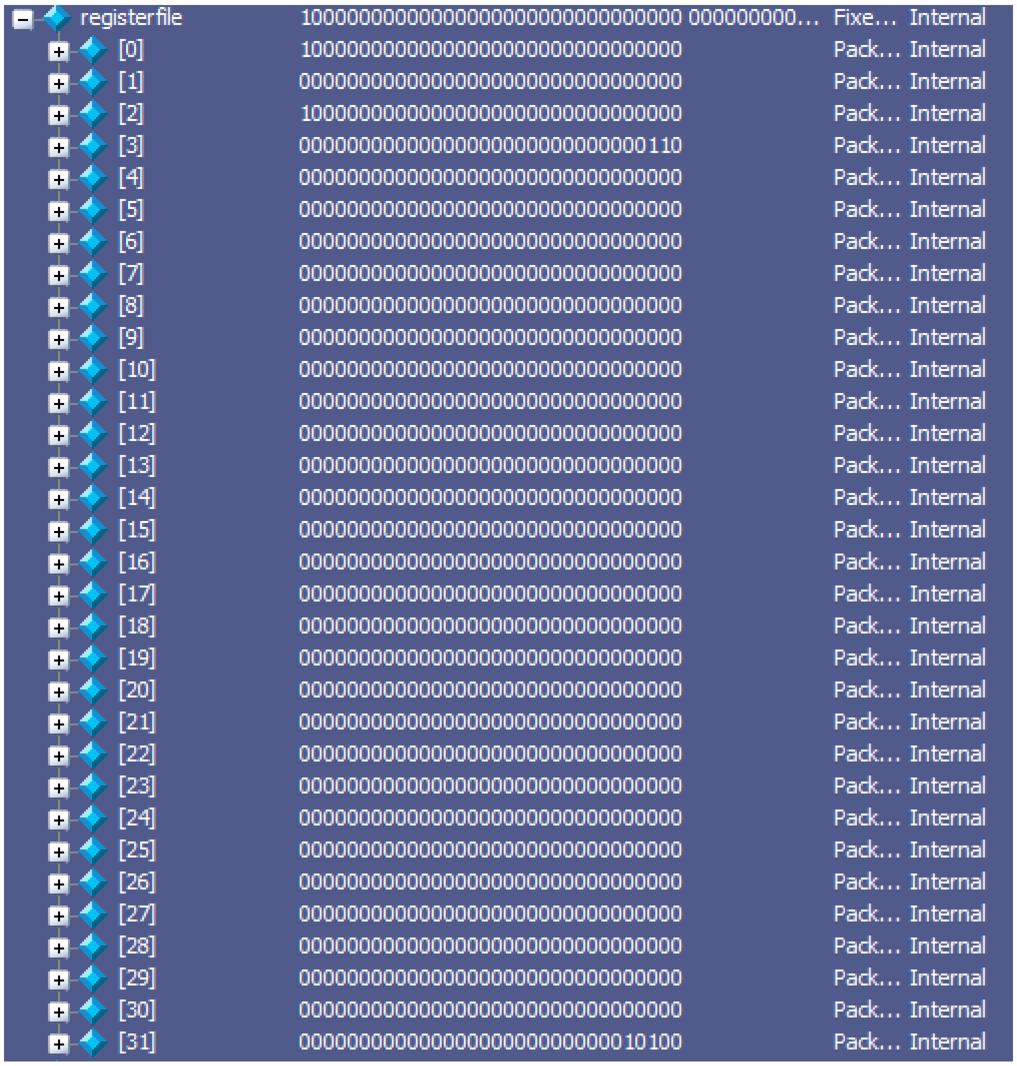
2.6.1 Test Case



```
lw $0, 0($8) # 100011 01000 00000 0000000000000000
lw $1, 4($8) # 100011 01000 00001 0000000000000100
add $2, $0, $1 # n flag 000000 00000 00001 00010 00000 100000
baln 24 # jumps to 6*4  011011 00000 00000 00000 00000 000110
add $6, $3, $6 # skipped 000000 00011 00110 00110 00000 100000
sw $7, 0($7)  # not skipped, memory[0] = 0 101011 00111 00111 0000000000000000

resultingRegister =
0 - x80000000
1 - x0
2 - x80000000
3 - x00000006
6 - 0
31 - x00000014
memory
0 - 0
```

Figure 15. Baln test case, and results in register and memory.

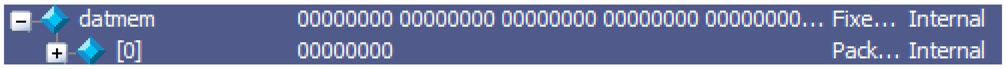
2.6.2 Register Result



```
registerfile  10000000000000000000000000000000 000000000...  Fixe...  Internal
  [0]         10000000000000000000000000000000               Pack...  Internal
  [1]         00000000000000000000000000000000               Pack...  Internal
  [2]         10000000000000000000000000000000               Pack...  Internal
  [3]         00000000000000000000000000000110               Pack...  Internal
  [4]         00000000000000000000000000000000               Pack...  Internal
  [5]         00000000000000000000000000000000               Pack...  Internal
  [6]         00000000000000000000000000000000               Pack...  Internal
  [7]         00000000000000000000000000000000               Pack...  Internal
  [8]         00000000000000000000000000000000               Pack...  Internal
  [9]         00000000000000000000000000000000               Pack...  Internal
  [10]        00000000000000000000000000000000               Pack...  Internal
  [11]        00000000000000000000000000000000               Pack...  Internal
  [12]        00000000000000000000000000000000               Pack...  Internal
  [13]        00000000000000000000000000000000               Pack...  Internal
  [14]        00000000000000000000000000000000               Pack...  Internal
  [15]        00000000000000000000000000000000               Pack...  Internal
  [16]        00000000000000000000000000000000               Pack...  Internal
  [17]        00000000000000000000000000000000               Pack...  Internal
  [18]        00000000000000000000000000000000               Pack...  Internal
  [19]        00000000000000000000000000000000               Pack...  Internal
  [20]        00000000000000000000000000000000               Pack...  Internal
  [21]        00000000000000000000000000000000               Pack...  Internal
  [22]        00000000000000000000000000000000               Pack...  Internal
  [23]        00000000000000000000000000000000               Pack...  Internal
  [24]        00000000000000000000000000000000               Pack...  Internal
  [25]        00000000000000000000000000000000               Pack...  Internal
  [26]        00000000000000000000000000000000               Pack...  Internal
  [27]        00000000000000000000000000000000               Pack...  Internal
  [28]        00000000000000000000000000000000               Pack...  Internal
  [29]        00000000000000000000000000000000               Pack...  Internal
  [30]        00000000000000000000000000000000               Pack...  Internal
  [31]        00000000000000000000000000010100               Pack...  Internal
```

Figure 16. The register file after Baln

2.6.3 Memory Result



```
datmem  00000000 00000000 00000000 00000000 00000000...  Fixe...  Internal
  [0]   00000000                                         Pack...  Internal
```

Figure 17. The data memory after Baln

3. Modifications to Datapath

3.1 Datapath Enhancements

ALU Modifications: As have been explained in the “2.” part of this report, as a summary, the ALU has been enhanced to support a NOR operation and handle immediate values through zero extension. New flags for zero, negative, and overflow conditions have been introduced to support conditional operations. A status register and a zero-extension component has been made to incorporate these into the datapath.

3.2 New Multiplexers

The names of the new multiplexers are MUX 7 through 11.

MUX 7: Routes either "Read Data 1" or a "Jump Address" to subsequent stages based on the balrnv control signal.

MUX 8: Selects between the data read from memory or the output of MUX 1, controlled by jmnor or jspal.

MUX 9: Chooses between "Read Data 2" or pc+4 based on jspal, affecting the data written back to memory or registers.

MUX 9, 10 & 11: These multiplexers manage register file operations, influencing which registers are read from or written to, under the control of jspal and bltzal.

3.3 Control Logic Enhancements

Control Signal Integration: As also have been explained in the “2.” part of this report, the new control signals including ori, bltzal, baln, jspal have been integrated into the control file to manage the flow of data and instructions based on the operation being performed.

Modifications to the control unit ensure that the correct paths are activated in the datapath, aligning execution with the intended instruction semantics.