EE446 Laboratory Work 5
Control Unit Design for Multi-Cycle CPU

Author: Ali Necat Karakuloğlu Date: 11/05/2020

ID: 2166742

Introduction

In this report, designed instruction set architecture will be stated and design will be defended. Based on ISA, the controller design will be explained and the test results for the required subroutines will be provided.

Instruction Set Architecture

This ISA is designed as 16-bit words that can support direct addressing, immediate addressing and indirect addressing modes according to the instructions. In this ISA, 8 registers are used. Furthermore, a program counter and a link register is used to achieve the requirements.

Instructions

As stated in the manual, there are 5 types of operation. Arithmetic, Logic, Shift, Branch and Memory. Rd denotes destination register, Rn denotes 1st operand and Rm denotes 2nd operand. Also, $Mem denotes a data in a memory address.

1.Arithmetic Operations

MNEMONIC	OPERATION	DESCRIPTION
ADD Rd, Rn, Rm	Rd = Rn + Rm	Register Addition
ADDI Rd, Rn, $Mem	Rd = Rn + $Mem	Indirect Addition with 4 bit Memory Addressing
SUB Rd, Rn, Rm	Rd = Rn - Rm	Register Addition
SUBI Rd, Rn, $Mem	Rd = Rn - $Mem	Indirect Addition with 4 bit Memory Addressing

2.Logic Operations

MNEMONIC	OPERATION	DESCRIPTION
AND Rd, Rn, Rm	Rd = Rn & Rm	Bitwise AND
ORR Rd, Rn, Rm	Rd = Rn | Rm	Bitwise OR
XOR Rd, Rn, Rm	Rd = Rn ^ Rm	Bitwise XOR
CLR Rd	Rd = 0	Clear

3.Shift Operations

MNEMONIC	OPERATION	DESCRIPTION
ROL Rd, Rn, Rm	Rd = {Rn[15-(Rm):(Rm)] , Rn[(Rm)-1:0]}	Rotate Circular Left
ROR Rd, Rn, Rm	Rd = {Rn[(Rm)-1:0] , Rn[15:(Rm)]}	Rotate Circular Right
LSL Rd, Rn, Rm	Rd = Rn << Rm	Logical Shift Left
ASR Rd, Rn, Rm	Rd = Rn <<< Rm	Arithmetic Shift Right
LSR Rd, Rn, Rm	Rd = Rn << Rm	Logical Shift Right

**(Rm) denotes value in Rm*

4.Branch Operations

MNEMONIC	OPERATION	DESCRIPTION
B Label	PC <= Label	Branch Uncond.
BL Label	PC <= Label , LR <= PC	Branch Uncond. With Link
BIL $Mem	PC <= $Mem , LR <= PC	Branch Indirect With Link
BIZ Label	PC <= Label	Branch If Zero
BNZ Label	PC <= Label	Branch If Not Zero
BIC Label	PC <= Label	Branch If Carry Set
BNC Label	PC <= Label	Branch If Carry Not Set
BLR	PC <= LR	Branch to LR Address

5.Logic Operations

MNEMONIC	OPERATION	DESCRIPTION
LDR Rd, $Mem	Rd = $Mem	Load to Register from Memory
MDR Rn, DATA	Rn = Data	Load Immediate Data to Register
SDM Rn, $Mem	$Mem = Rn	Store from Register to Memory

Instruction Format

In this instruction format op represents the operations i.e. arithmetic, logic etc. and ops field represents instructions in an operation. Extension field is clarified in the following part of this section.

OP	OPS	INSTRUCTION	EXTENSION
000(ARITHMETIC)	000	ADD	Rd(3-bit), Rn(3-bit), Rm(3-bit)
000(ARITHMETIC)	001	ADDI	Rd(3-bit),Rn(3-bit), $Mem(4-bit)
000(ARITHMETIC)	010	SUB	Rd(3-bit), Rn(3-bit), Rm(3-bit)
000(ARITHMETIC)	011	SUBI	Rd(3-bit),Rn(3-bit), $Mem(4-bit)
001(LOGIC)	000	AND	Rd(3-bit), Rn(3-bit), Rm(3-bit)
001(LOGIC)	001	OR	Rd(3-bit), Rn(3-bit), Rm(3-bit)
001(LOGIC)	010	XOR	Rd(3-bit), Rn(3-bit), Rm(3-bit)
001(LOGIC)	011	CLR	Rd(3-bit), Rn(3-bit), Rm(3-bit)
010(SHIFT)	000	ROL	Rd(3-bit), Rn(3-bit), Shift Amt.(4-bit)
010(SHIFT)	001	ROR	Rd(3-bit), Rn(3-bit), Shift Amt.(4-bit)
010(SHIFT)	010	LSL	Rd(3-bit), Rn(3-bit), Shift Amt.(4-bit)
010(SHIFT)	011	ASR	Rd(3-bit), Rn(3-bit), Shift Amt.(4-bit)
010(SHIFT)	100	LSR	Rd(3-bit), Rn(3-bit), Shift Amt.(4-bit)
011(BRANCH)	000	B	Label(10-bit)
011(BRANCH)	001	BL	Label(10-bit)
011(BRANCH)	010	BIL	3-bit 0 , $Mem(7-bit)
011(BRANCH)	011	BIZ	Label(10-bit)
011(BRANCH)	100	BNZ	Label(10-bit)
011(BRANCH)	101	BIC	Label(10-bit)
011(BRANCH)	110	BNC	Label(10-bit)
011(BRANCH)	111	BLR	-
100(MEMORY)	000	LDR	Rd(3-bit), $Mem(7-bit)
100(MEMORY)	001	MDR	Rd(3-bit), DATA(7-bit)
100(MEMORY)	010	SDM	Rn(3-bit), $Mem(7-bit)

Further Clarifications on the ISA Design

Note that in the datapath design task, the CPU ought to have at least 8 registers. To represent 8 registers, 3 bits are required. Therefore, in an arithmetic operation the CPU need to have the information of source, first and second operands. In total, 9 bits are required to represent registers. Furthermore, 24 instructions have to be represented. Thus, 5 bits are required. In total 14 bits are required. In order to represent the instructions in hexadecimal easily and extend the range of branch instructions and indirect memory read range, instructions are 16 bits. There tradeoffs in this design as all the other designs may have. First one is, the memory is not byte addressable. The reason is, indirect addresses has a small range. If it was byte addressable, the memory capacity would be smaller. Another tradeoff is direct addressing is not supported except in MDR instruction. However, one can load any 16 bit word to the registers with using MDR and LSL operations.

Multicycle CPU Controller Unit Design

As mentioned in the datapath design, the controller activates some multiplexer select signals and register enable signals. In this part of the report the details of each state will be given with the corresponding signals. The design is a 2 stage FSM that has one combinational part and a sequential part that controls the flow of operations and signals. In other words, no modular design is made as in ARM architecture in the lecture notes. Decoding of the operations are not made in a separate unit. Each signal and next states are defined by conditional statements that depends on the corresponding instruction fields. Abstract fetch, decode and execute stages will be explained with their states in the asm chart.

Fetch

In this state, PC address is read by memory and PC+1 calculation is made by using ALU. The next state is decode. If the CPU is reset, this will be the state where the controller start. As can be seen from the figure 1, in this phase, only fetch state exists.

Corresponding States: FETCH

Decode

This state takes 2 clock cycles because, the CPU must wait until instruction register is loaded. Note that, in fetch the memory is just read by PC. The first cycle is just to wait until instruction register is written. Next state is determined by instruction fields. There are 3 states, first one is memory, second one is branch and third one is ALOP (Arithmetic Logic OPerations). In those states, PC and register file buffers are written. If another memory read is required, it is made in memory read indirect state, in this state all other operations are halted and an extra memory read is made according to the instruction.

Corresponding States: DECODE, DECODE_MEM, DECODE_BIL_BL, DECODE_ALOP_REGISTER, MEM_READ_IND_ALOP

Execute

In this state, alu operations and register-memory write operations are done. As it was done in the previous parts, alu control signals and the select signals are determined according to the instruction fields. Note that the most significant 6 bits are the important ones. For example, add operation sends alu ctr as 000 and connects RD1 and RD2 to inputs of alu. Then, alop writeback state is selected as the next state. However, as can be seen from the figure 1, if an operation is memory or branch, alu operations aren't needed thus execute alop state is skipped.

Corresponding States:

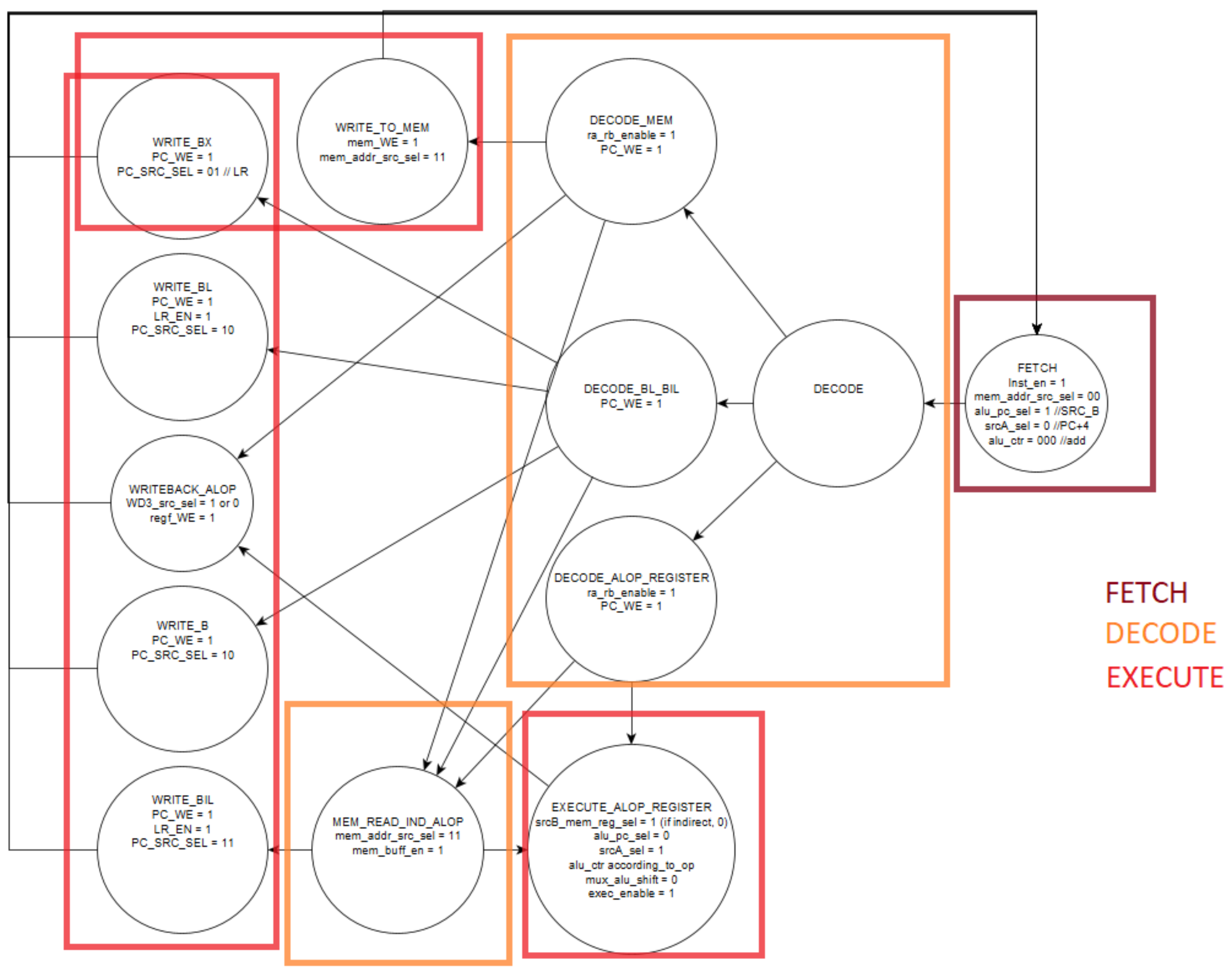
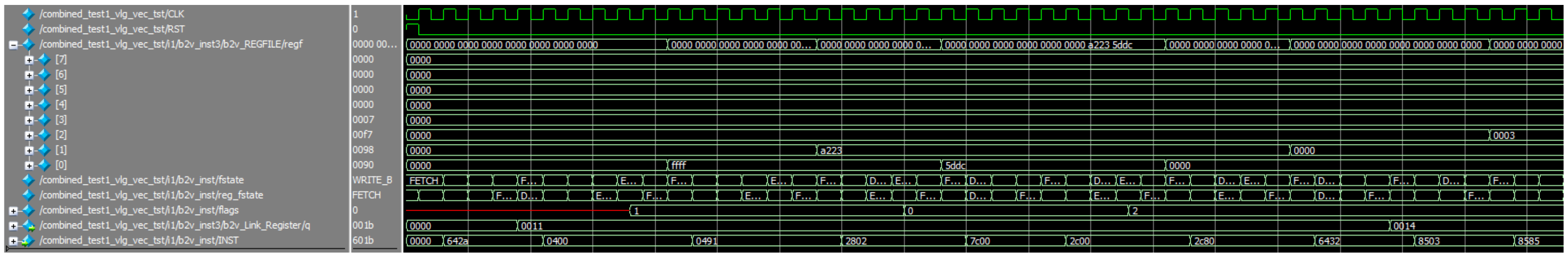


Figure 1: ASM Chart of the Controller Unit

Test of the Multicycle CPU

In this part, test results and machine codes are provided. The testbench only provides clock and reset signals.

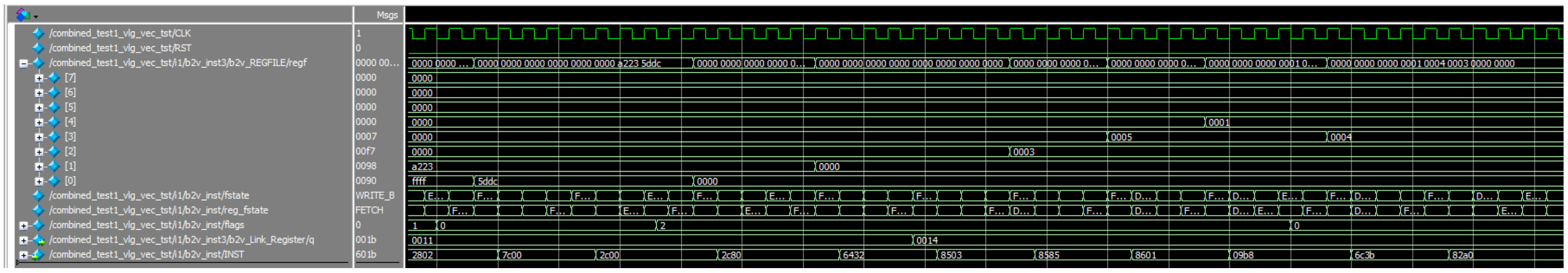
As can be seen from the figure 2, the subroutine complements the number given in memory location 1. The result can be seen in R0 register as “5dcc” before “7c00” instruction. And after the subroutine, PC returns to main.



```
/combined_test1_vlg_vec_tst/CLK                                   1
/combined_test1_vlg_vec_tst/RST                                   0
/combined_test1_vlg_vec_tst/i1/b2v_inst3/b2v_REGFILE/regf         0000 00...
  [7]                                                             0000
  [6]                                                             0000
  [5]                                                             0000
  [4]                                                             0000
  [3]                                                             0007
  [2]                                                             00f7
  [1]                                                             0098
  [0]                                                             0090
/combined_test1_vlg_vec_tst/i1/b2v_inst/fstate                    WRITE_B
/combined_test1_vlg_vec_tst/i1/b2v_inst/reg_fstate                FETCH
/combined_test1_vlg_vec_tst/i1/b2v_inst/flags                     0
/combined_test1_vlg_vec_tst/i1/b2v_inst3/b2v_Link_Register/q      001b
/combined_test1_vlg_vec_tst/i1/b2v_inst/INST                      601b
```

Figure 2: Test Results of the 1st Subroutine

As can be seen from the figures 3a, 3b, 3c and 3d, the subroutine sums the elements of the array in memory. The result can be seen in R0 register as “0170” before “7c00” instruction. And after the subroutine, PC returns to main, clears registers. Because of the limited space, results are given separately.



```
                                                                  Msgs
/combined_test1_vlg_vec_tst/CLK                                   1
/combined_test1_vlg_vec_tst/RST                                   0
/combined_test1_vlg_vec_tst/i1/b2v_inst3/b2v_REGFILE/regf         0000 00...
  [7]                                                             0000
  [6]                                                             0000
  [5]                                                             0000
  [4]                                                             0000
  [3]                                                             0007
  [2]                                                             00f7
  [1]                                                             0098
  [0]                                                             0090
/combined_test1_vlg_vec_tst/i1/b2v_inst/fstate                    WRITE_B
/combined_test1_vlg_vec_tst/i1/b2v_inst/reg_fstate                FETCH
/combined_test1_vlg_vec_tst/i1/b2v_inst/flags                     0
/combined_test1_vlg_vec_tst/i1/b2v_inst3/b2v_Link_Register/q      001b
/combined_test1_vlg_vec_tst/i1/b2v_inst/INST                      601b
```

Figure 3a: Test Results of the 2nd Subroutine

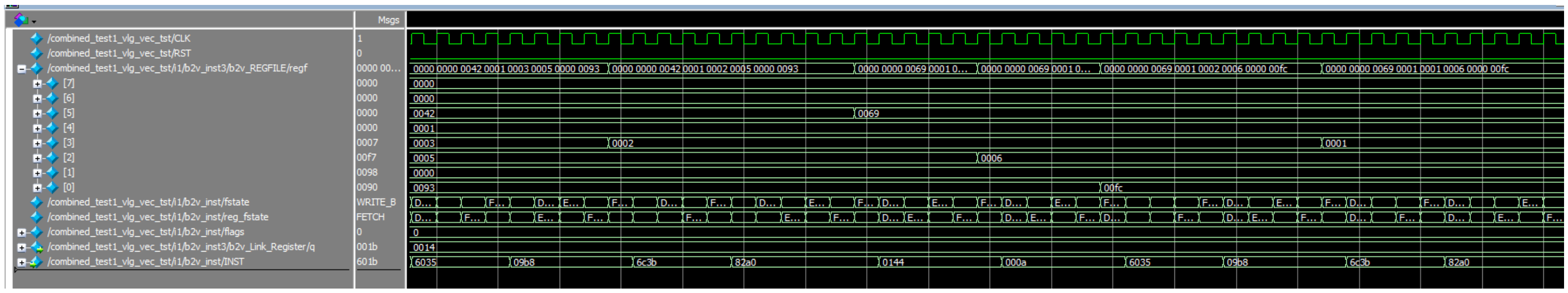

Msgs
/combined_test1_vlg_vec_tst/CLK
/combined_test1_vlg_vec_tst/RST
/combined_test1_vlg_vec_tst/i1/b2v_inst3/b2v_REGFILE/regf
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
/combined_test1_vlg_vec_tst/i1/b2v_inst/fstate
/combined_test1_vlg_vec_tst/i1/b2v_inst/reg_fstate
/combined_test1_vlg_vec_tst/i1/b2v_inst/flags
/combined_test1_vlg_vec_tst/i1/b2v_inst3/b2v_Link_Register/q
/combined_test1_vlg_vec_tst/i1/b2v_inst/INST

Figure 3b: Test Results of the 2nd Subroutine

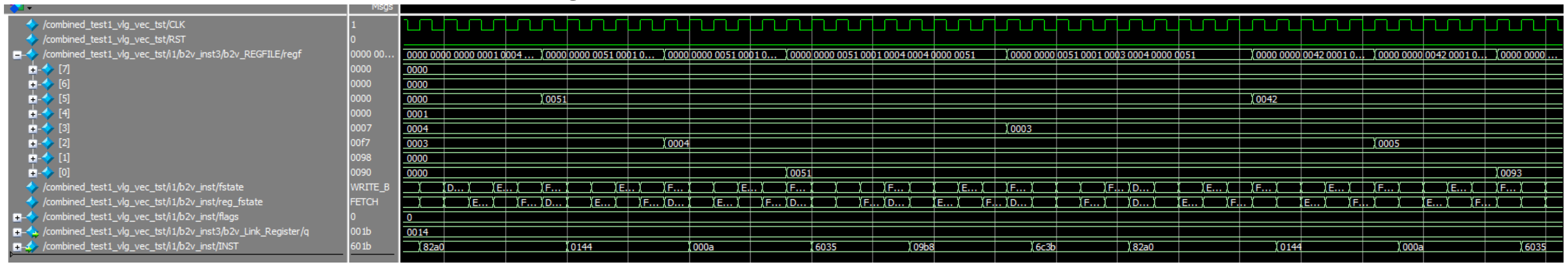

Msgs
/combined_test1_vlg_vec_tst/CLK
/combined_test1_vlg_vec_tst/RST
/combined_test1_vlg_vec_tst/i1/b2v_inst3/b2v_REGFILE/regf
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
/combined_test1_vlg_vec_tst/i1/b2v_inst/fstate
/combined_test1_vlg_vec_tst/i1/b2v_inst/reg_fstate
/combined_test1_vlg_vec_tst/i1/b2v_inst/flags
/combined_test1_vlg_vec_tst/i1/b2v_inst3/b2v_Link_Register/q
/combined_test1_vlg_vec_tst/i1/b2v_inst/INST

Figure 3c: Test Results of the 2nd Subroutine

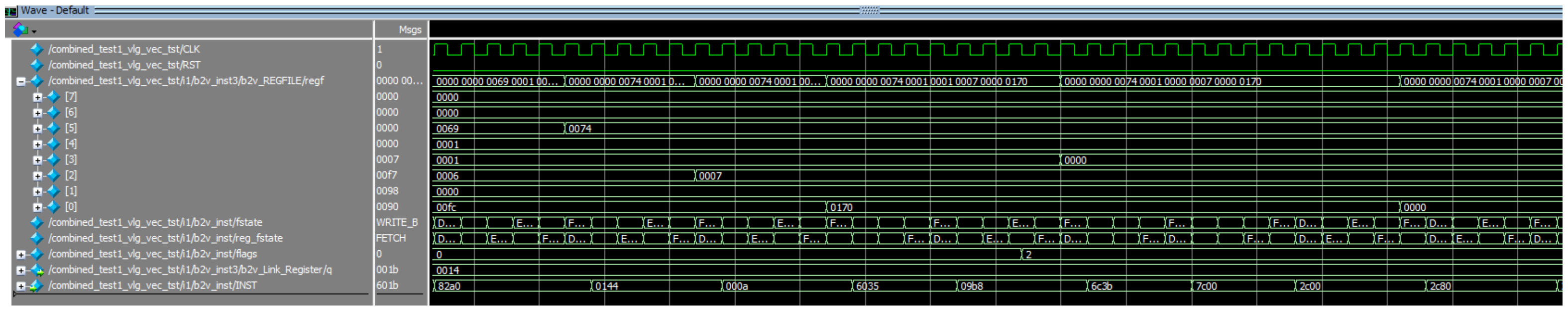

Wave - Default
Msgs
/combined_test1_vlg_vec_tst/CLK
/combined_test1_vlg_vec_tst/RST
/combined_test1_vlg_vec_tst/i1/b2v_inst3/b2v_REGFILE/regf
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
/combined_test1_vlg_vec_tst/i1/b2v_inst/fstate
/combined_test1_vlg_vec_tst/i1/b2v_inst/reg_fstate
/combined_test1_vlg_vec_tst/i1/b2v_inst/flags
/combined_test1_vlg_vec_tst/i1/b2v_inst3/b2v_Link_Register/q
/combined_test1_vlg_vec_tst/i1/b2v_inst/INST

Figure 3d: Test Results of the 2nd Subroutine

As can be seen from the figures 4, the subroutine determines whether the number is odd or even and changes the order of the bits according to the specifications given in the manual. The result can be seen in R0 register as “0090” before “7c00” instruction. And after the subroutine, PC returns to main, clears registers. The program finishes with an infinite loop,

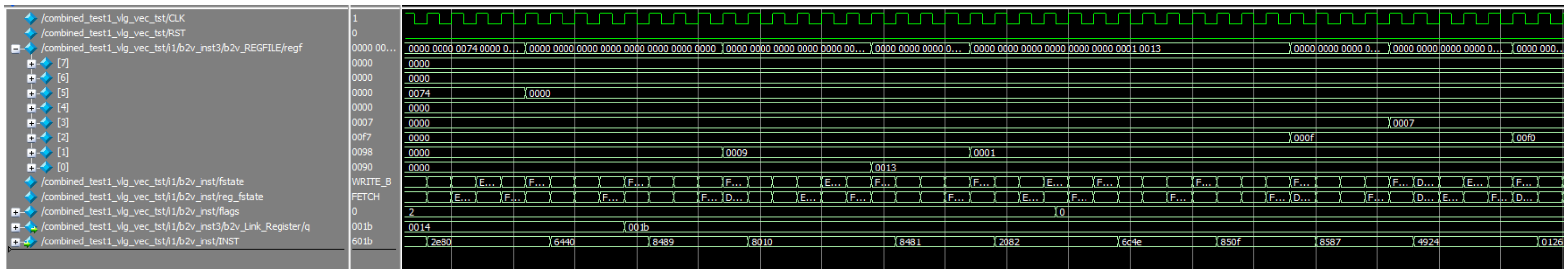

```
/combined_test1_vlg_vec_tst/CLK                         1
/combined_test1_vlg_vec_tst/RST                         0
/combined_test1_vlg_vec_tst/i1/b2v_inst3/b2v_REGFILE/regf   0000 00...
  [7]                                                   0000
  [6]                                                   0000
  [5]                                                   0000
  [4]                                                   0000
  [3]                                                   0007
  [2]                                                   00f7
  [1]                                                   0098
  [0]                                                   0090
/combined_test1_vlg_vec_tst/i1/b2v_inst/fstate          WRITE_B
/combined_test1_vlg_vec_tst/i1/b2v_inst/reg_fstate      FETCH
/combined_test1_vlg_vec_tst/i1/b2v_inst/flags           0
/combined_test1_vlg_vec_tst/i1/b2v_inst3/b2v_Link_Register/q   001b
/combined_test1_vlg_vec_tst/i1/b2v_inst/INST            601b
```

Figure 4a: Test Results of the 3rd Subroutine

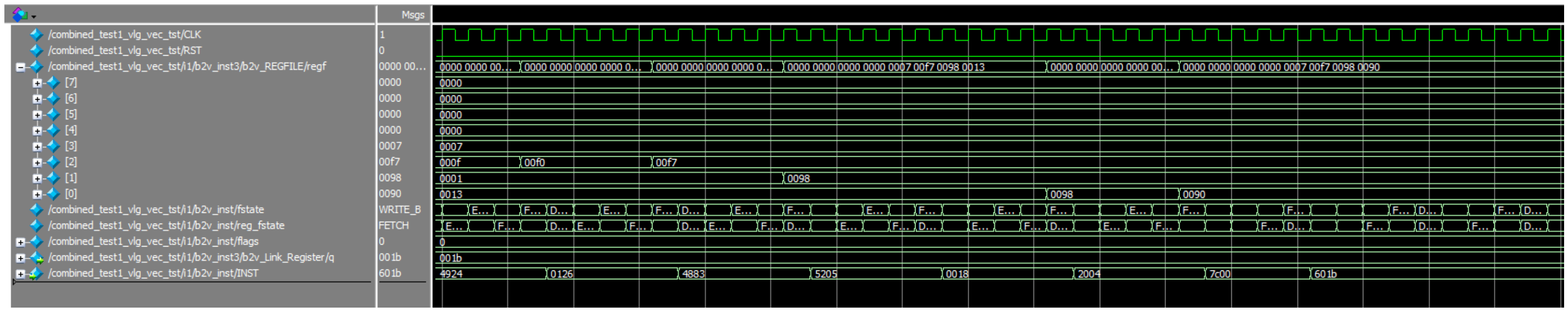

```
                                                        Msgs
/combined_test1_vlg_vec_tst/CLK                         1
/combined_test1_vlg_vec_tst/RST                         0
/combined_test1_vlg_vec_tst/i1/b2v_inst3/b2v_REGFILE/regf   0000 00...
  [7]                                                   0000
  [6]                                                   0000
  [5]                                                   0000
  [4]                                                   0000
  [3]                                                   0007
  [2]                                                   00f7
  [1]                                                   0098
  [0]                                                   0090
/combined_test1_vlg_vec_tst/i1/b2v_inst/fstate          WRITE_B
/combined_test1_vlg_vec_tst/i1/b2v_inst/reg_fstate      FETCH
/combined_test1_vlg_vec_tst/i1/b2v_inst/flags           0
/combined_test1_vlg_vec_tst/i1/b2v_inst3/b2v_Link_Register/q   001b
/combined_test1_vlg_vec_tst/i1/b2v_inst/INST            601b
```

Figure 4b: Test Results of the 3rd Subroutine

```
MACHINE     LABEL       INSTRUCTION
CODE
***************************************
642A                MAIN        BL      COMPLEMENT
2C00                                    CLR R0
2C80                                    CLR R1
6432                                    BL          ARRAY_SUM
2C00                                    CLR R0
2C80                                    CLR R1
2D00                                    CLR R2
2D80                                    CLR R3
2E00                                    CLR R4
2E80                                    CLR R5
6440                                    BL  ODD_EVEN
601B                END                 B           END
0400                COMPLEMENT          ADDI R0, $0 ;$0 contains all 1(FFFF)
0491                                    ADDI R1, $1 ;$1 contained the number to be complemented
2802                                    XOR  R1,R1,R0; complement is in R1
7C00                                    BLR                     ;Branch to link register
8503                ARRAY_SUM           MDR  R2, #3; Array Pointer is loaded to R2
8585                                    MDR  R3, #5; Array count
8601                                    MDR  R4, #1;
09B8                LOOP                SUB  R3,R3,R4
6C3B                                    BIZ  LOOP_END ; branch to loop end if r3 is zero
82A0                                    LDR  R5,[R2]
0144                                    ADD  R2, R2, R4; Increment pointer
000A                                    ADD  R0, R0, R5; accumulate in r0
6035                                    B   LOOP
7C00                LOOP_END            BLR
8489                ODD_EVEN            MDR R1,#9   ;load the address of number
8010                                    LDR R0,[R1] ;load number in r0
8481                                    MDR R1,#1   ;load r1 1
2082                                    AND R1,R0,R1;
6C4E                                    BIZ EVEN    ;If zero branch to even
850F                ODD                 MDR R2, #F
8587                                    MDR R3, #7
4924                                    LSL R2, #4
0126                                    ADD R2, R3, R2 ;f7 is loaded to r2
4883                                    LSL R1, R0, #3 ;shift 3 times to place the digits in the right place
5205                                    LSR R4, R0, #5
0018                                    ADD R1, R4,R1
2004                                    AND R0, R2, R0; Result is in r0
7C00                                    BLR
8507                EVEN                MDR R2, #7
8588                                    MDR R3, #8
4924                                    LSL R2, #4
0126                                    ADD R2, R3, R2 ;78 is loaded to r2
4802                                    LSL R0,R0,#2
2004                                    AND R0,R0,R2
7C00                                    BLR
```

Note: Memory content can be seen from new_mem.mif file. Especially check the first 16 addresses.

Note 2: Controller is also explained in the code, it may be helpful to go with the code and instruction table to understand easier.

Conclusion

In this report, the instruction set is clarified, its tradeoffs and other features are explained. Then, the controller design is explained and required subroutines are demonstrated with the simulation results.