Kintel – j8

40 MHz 8-BIT CPU DESIGN REPORT
EEE-446

BARIŞ ÇAĞRI DİNÇ

İçindekiler Tablosu

Introduction

To become computer architects, we are given our first objective designing a CPU from ground up. To achieve this we started from basic algorithm implementations to full control unit design and datapath design. So, to explain our work in this report we will introduce our CPU design, Kintel j8.

As a project of Computer Architecture course, we designed an 8-bit CPU capable of array computation, text editing, signed multiplication and division. We selected multi-cycle work flow for our instructions. ISA design is made based on benchmarks provided to us. Design metrics, performance of CPU can be found in this report with full design codes, diagrams and tables. This design implemented on Cyclone II FPGA on the board of Terasic's DE2-70 board. Design software used is Quartus II v13.0.

ISA Design

Instruction Types

In this design, we assigned 4 bits for the OpCode because there are 15 instructions in the Kintel – j8 ISA. We tried to keep instruction number minimal to reduce complexity. However, further decrease cannot be achieved because benchmark programs require 15 instructions to satisfy also conditions of the further lab exercises and different addressing modes need at least that much number of instructions. Kintel – j8 have 6 different address modes which are load byte, store byte, store upper byte, load upper byte. There are different modes of stores and loads instructions to reach individual bytes in the memory to achieve memory usage performance to maximum. In addition, Kintel – j8 has immediate instruction to load an 8-bit value to a register directly because we cannot achieve 8-bit load with other instructions. Also, ALU operations, Booth Multiplier, and Division outputs are 8-bit. There are two types of branch instruction and one type of Jump instruction to change the Program Counter state. Also, we have an instruction which is Set if Less Than. By using this instruction different type of Pseudo Branch instructions can be created like Branch if greater than or Branch if less than. In addition, there is jump instruction, which can change Program Counter state directly. Main aim to achieve is running benchmark programs on this ISA. Further improvements for this ISA require more bits for instructions that increases complexity. Finally, Kintel – j8 instruction set architecture created to be small, simple, and fixed for common cases to reduced complexity and reach high speed and low cost and power.

R – TYPE

OPCODE	SrCAdrA	SrCAdrB	DstAdr0	DstAdr1
4-bit	3-bit	3-bit	3-bit	3-bit

I – TYPE

OPCODE	SrcBdrB(destination)	SrcBdrA	IMMEDIATE
4-bit	3-bit	3-bit	6-bit

J – TYPE

OPCODE	IMMEDIATE
4-bit	12-bit

IM – TYPE

OPCODE	DstAdr	IMMEDIATE	--
4-bit	3-bit	8-bit	2-bit

OPERATION	DESCP.	ASSEMBLY	TYPE	OPCODE
ADD	**Addition**	ADD DstAdr0, SrcAdrA, SrcAdrB	**R**	**0000**
SUB	**Subtraction**	SUB DsrAdr0, SrcAdrA, SrcAdrB	**R**	**0001**
MULT	**Multiplication**	MULT DstAdr0, DstAdr1, SrcAdrA, SrcAdrB	**R**	**0010**
DIV	**Division**	DIV DstAdr0, DstAdr1, SrcAdrA, SrcAdrB	**R**	**0011**
BEQ	**Branch if equal**	BEQ SrcAdrB , SrcAdrA, LABEL	**I**	**0100**
BNEQ	**Branch if not equal**	BNEQ SrcAdrB , SrcAdrA, LABEL	**I**	**0101**
JMP	**Jump**	JMP IMMEDIATE	**J**	**0110**
SB	**Store byte**	SB SrcAdrB , IMMEDIATE(SrcAdrA)	**I**	**0111**
LB	**Load byte**	LB SrcAdrB, IMMEDIATE(SrcAdrA)	**I**	**1000**
MOVI	**Move Immediate to register**	MOVI DstAdr , IMMEDIATE	**IM**	**1010**
ADDI	**Add immeadiate**	ADDI SrcAdrB, SrcBdrA, IMMEDIATE	**I**	**1011**
SUBBI	**Subtract immediate**	SUBBI SrcAdrB, SrcBdA, IMMEDIATE	**I**	**1100**
SLT	**Set if less than**	SLT DstAdr0, SrcAdrA, SrcAdrB	**R**	**1101**
LBU	**Load upper byte**	LBU SrcAdrB, IMMEDIATE(SrcBdrA)	**I**	**1110**
SBU	**Store upper byte**	SBU SrcAdrB, IMMEDIATE(SrcBdrA)	**I**	**1111**

Instruction Descriptions

1)

ADD DsrAdr0, SrcAdrA, SrcAdrB → R-type instruction

OpCode = 0000

DsrAdr0 = SrcAdrA + SrcAdrB PC = PC + 1

Ex : ADD $1, $2, $3

0000	010	011	001	000

2)

SUB DsrAdr0, SrcAdrA, SrcAdrB → R-type instruction

OpCode = 0001

DsrAdr0 = SrcAdrA – SrcAdrB PC = PC + 1

Ex : SUB $1, $2, $3

0001	010	011	001	000

3)

MULT DsrAdr0, DsAdr1, SrcAdrA, SrcAdrB → R-type instruction

OpCode = 0010

DsrAdr0 , DsrAdr1 = SrcAdrA * SrcAdrB (8-bit * 8bit =16bits) That is why 2 different 8-bit registers used to hold 16-bit data

Ex : MULT $1, $2, $3 , $4 PC = PC + 1

0010	011	100	001	010

4)

DIV DsrAdr0, DstAdr1, SrcAdrA, SrcAdrB → R-type instruction
OpCode = 0011

DsrAdr0 = SrcAdrA / SrcAdrB PC = PC + 1

DstAdr1 = Remainder

Ex : ADD $1,$2 $3, $4

0011	011	100	001	010

5)

BEQ SrcAdrB , SrcAdrA , LABEL → I – type instruction

OpCode = 0100

IF SrcAdrA = SrcAdrB , PC = PC + 1 ± R(Relative Location)

0100	100	101	000111

6)

BNEQ SrcAdrB , SrcAdrA , LABEL → I – type instruction
Opcode = 0101

IF SrcAdrA != SrcAdrB , PC = PC + 1 ± R(Relative Location)

0101	100	101	000111

7)

JMP IMMEDIATE → J-type instruction
Opcode = 0110

PC = IMMEDIATE ADRESS

Ex: JMP IMMEDIATE → IMMEDIATE ADRESS = 0x100

0110	000100000000

8)

SB SrcAdrB , IMMEDIATE(SrcAdrA) → I-type (Load Byte 8-bit)
Opcode = 0111

MEM [IMMEDIATE(SrcAdrB)] = The content of SrcAdrA

MEM[IMMEDIATE + R[SrcAdrB]] = The content of SrcAdrA (index addressing)

Ex: SB $5, 3($6) PC = PC + 1

0111	101	110	000011

9)

LB SrcAdrA , IMMEDIATE(SrcAdrB) → I-type (Load Byte 8-bit)
Opcode = 1000

The content of SrcAdrA = MEM [IMMEDIATE(SrcAdrB)]

Ex: LB $5, 3($6) PC = PC + 1

1000	101	110	000011

10)

MOVI DstAdr , IMMEDIATE → IM-type (Load Immediate 8-bit)
Opcode = 1010

DstAdr = IMMEDIATE PC = PC + 1

Ex: MOVI $5, 10

1010	101	00001010	--

11)

ADDI SrcAdrB, SrcBdA, IMMEDIATE → I-type

OpCode = 1011

SrcAdrA = SrcAdrB + IMMEDIATE PC = PC + 1

Ex : ADDI $1,$2, 15

1011	001	010	001111

12)
SUBBI SrcAdrB, SrcBdA, IMMEDIATE → I-type

OpCode = 1100

SrcAdrA = SrcAdrB – IMMEDIATE PC = PC + 1

Ex : ADDI $1,$2, 15

1100	001	010	001111

13)

SLT DsrAdr0, SrcAdrA, SrcAdrB → R-type instruction

OpCode = 1101 (Set if less than)

If SrcAdra < SrcAdrB then set DsrAdr0 = 0xFF

Ex : ADD \$1, \$2, \$3 PC = PC + 1

1101	010	011	001	000

14)

LBU SrcAdrB, IMMEDIATE(SrcBdrA) → I – type instruction

OpCode = 1110

SrcAdrA= Content of Upper byte MEM[IMMEDIATE(SrcBdrB)]

Ex: LBU \$2, 5(\$3) PC = PC + 1

1110	010	011	000101

15)

SBU SrcAdrA, IMMEDIATE(SrcBdrB) → I – type instruction

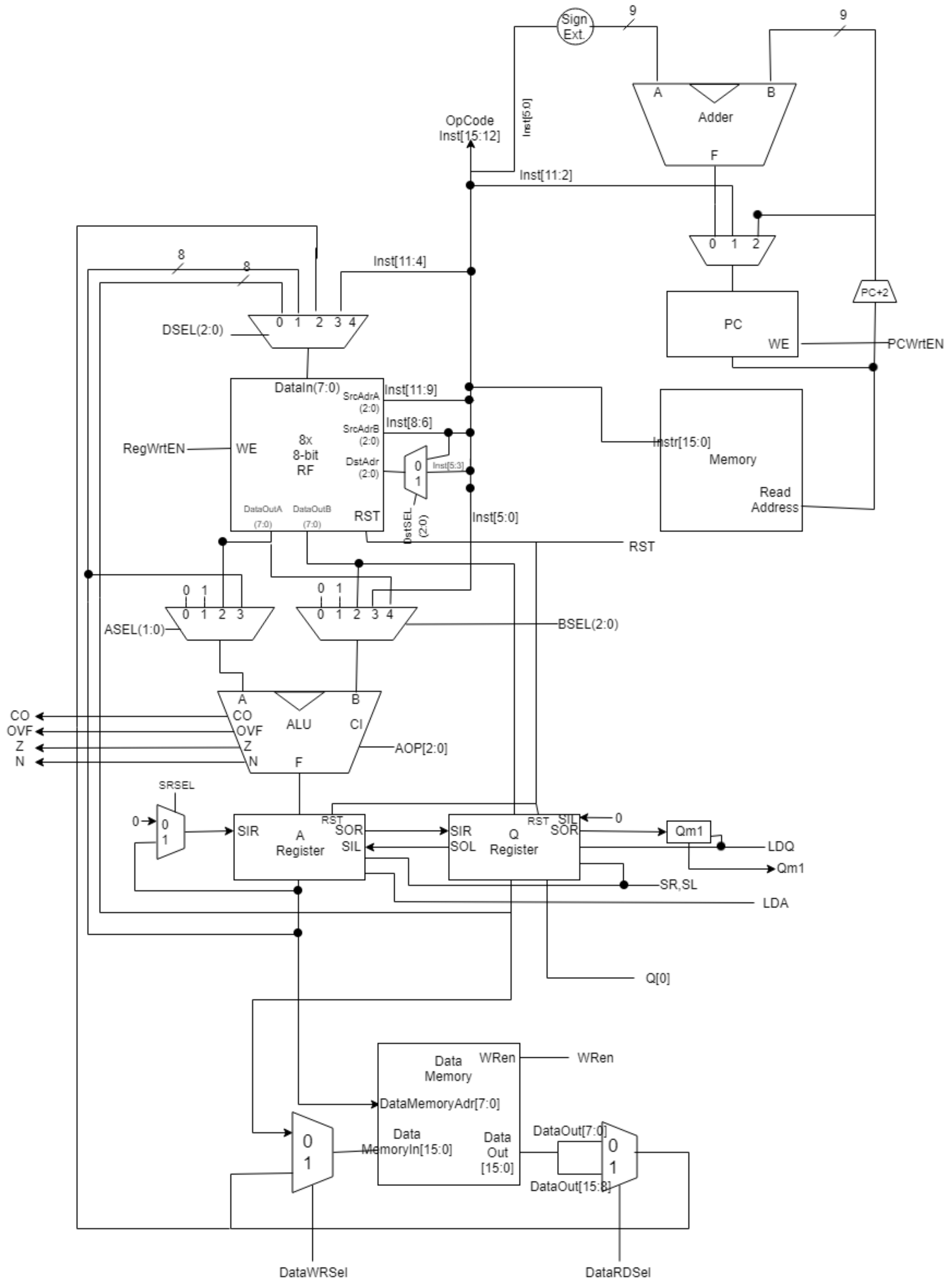
OpCode = 1111

Upper byte MEM[IMMEDIATE(SrcBdrB)] = SrcAdrA

Ex: SBU \$2, 5(\$3) PC = PC + 1

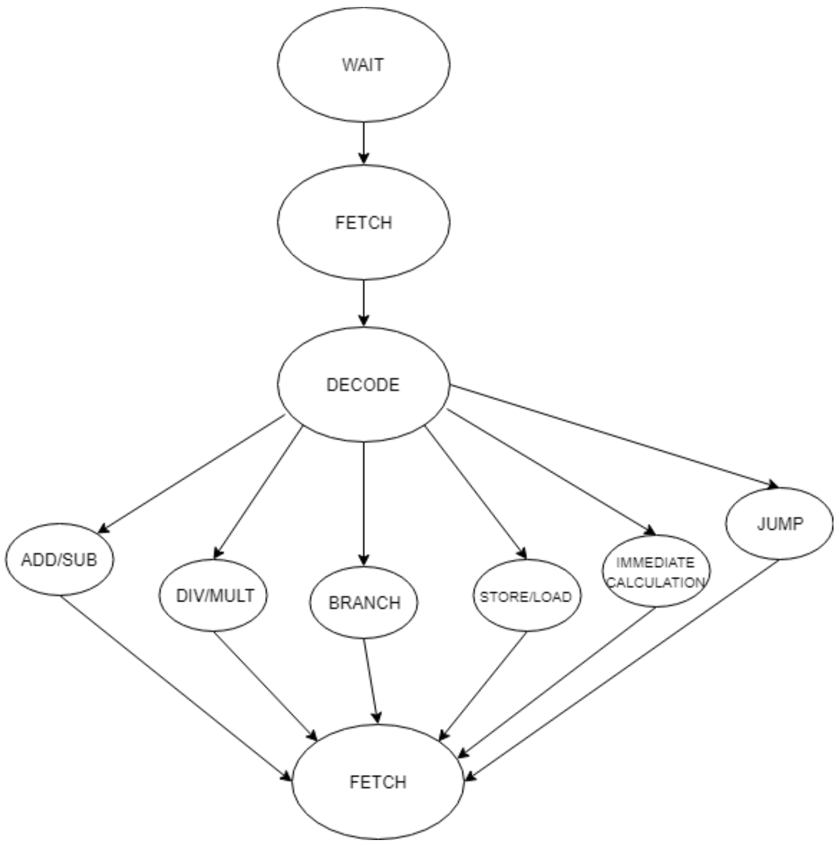
1111	010	011	000101

Datapath Design



Control Unit Design

Basic State Diagram can be found on figure below. One-Hot state names can be found below. With them wanted state signals can be found from VHDL code itself.

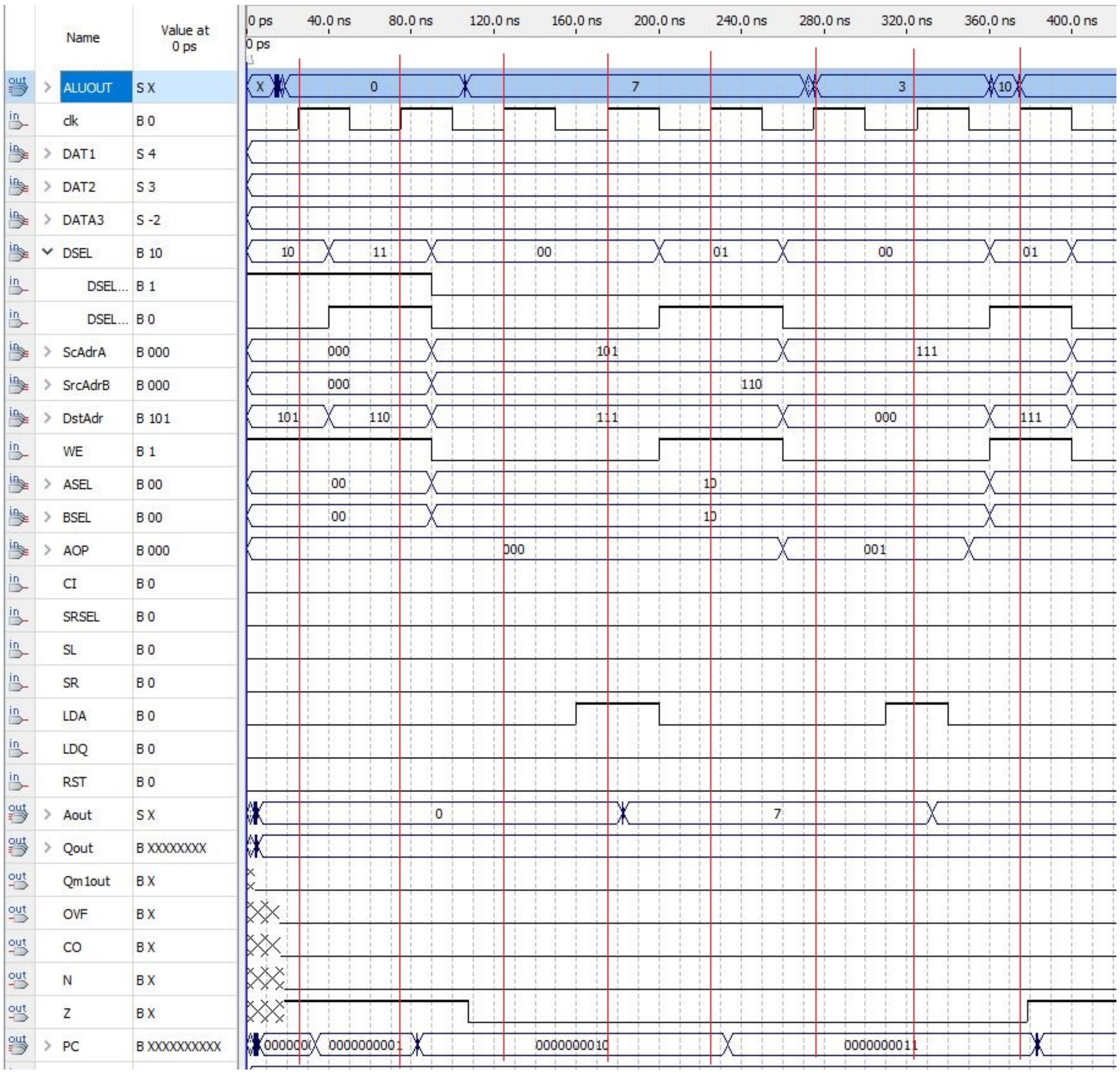


OPERATION	DESCP.	State Numbers	TYPE	OPCODE
ADD	Addition	S6 and S8(Writeback)	R	0000
SUB	Subtraction	S7 and S8(Writeback)	R	0001
MULT	Multiplication	S22,S23,S24,S25,S26(Writeback) and S27(Writeback)	R	0010
DIV	Division	S28,S29,S30,S31,S32 and S33(Writeback)	R	0011
BEQ	Branch if equal	S10,S12 andS13	I	0100
BNEQ	Branch if not equal	S11,S14 and S13	I	0101
JMP	Jump	S9	J	0110
SB	Store byte	S15 and S16	I	0111
LB	Load byte	S15 and S18	I	1000
MOVI	Move Immediate to register	S4	IM	1010
ADDI	Add immeadiate	S2 and S3(Writeback)	I	1011
SUBBI	Subtract immediate	S5 and S3(Writeback)	I	1100
SLT	Set if less than	S7,S20(Writeback) and S21(Writeback)	R	1101
LBU	Load upper byte	S15 and S19	I	1110
SBU	Store upper byte	S15 and S17	I	1111

Test and Simulations

Simulations

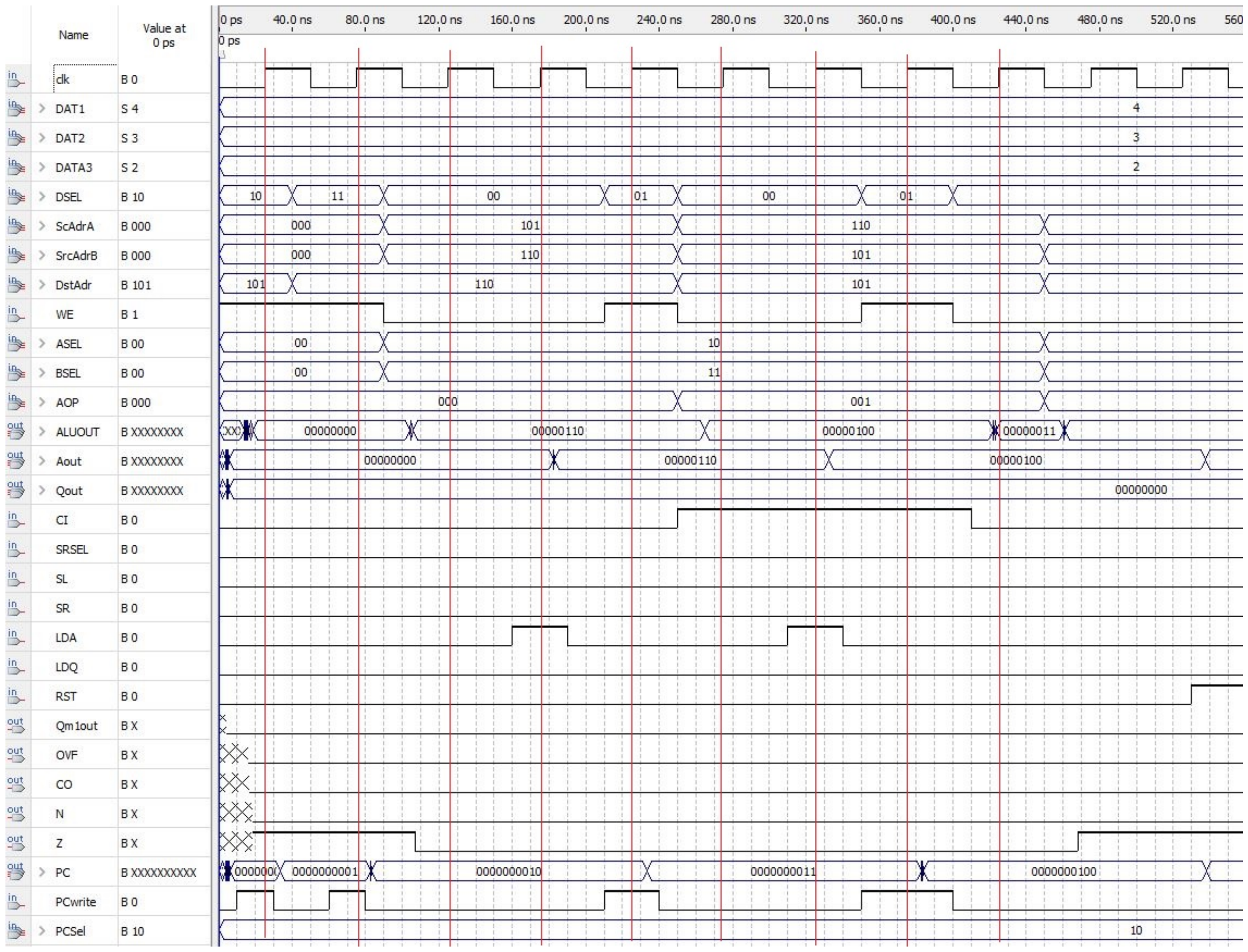
MOVI,ADD,SUB



```
MOVI $5, 4
MOVI $6, 3
ADD $7,$4,$5
SUB $7,$7,$6
```

Lines added to track every cycle, MOVI takes 1 cycle while ADD and SUB takes 3 cycles. While PC increases 1 for each instruction.

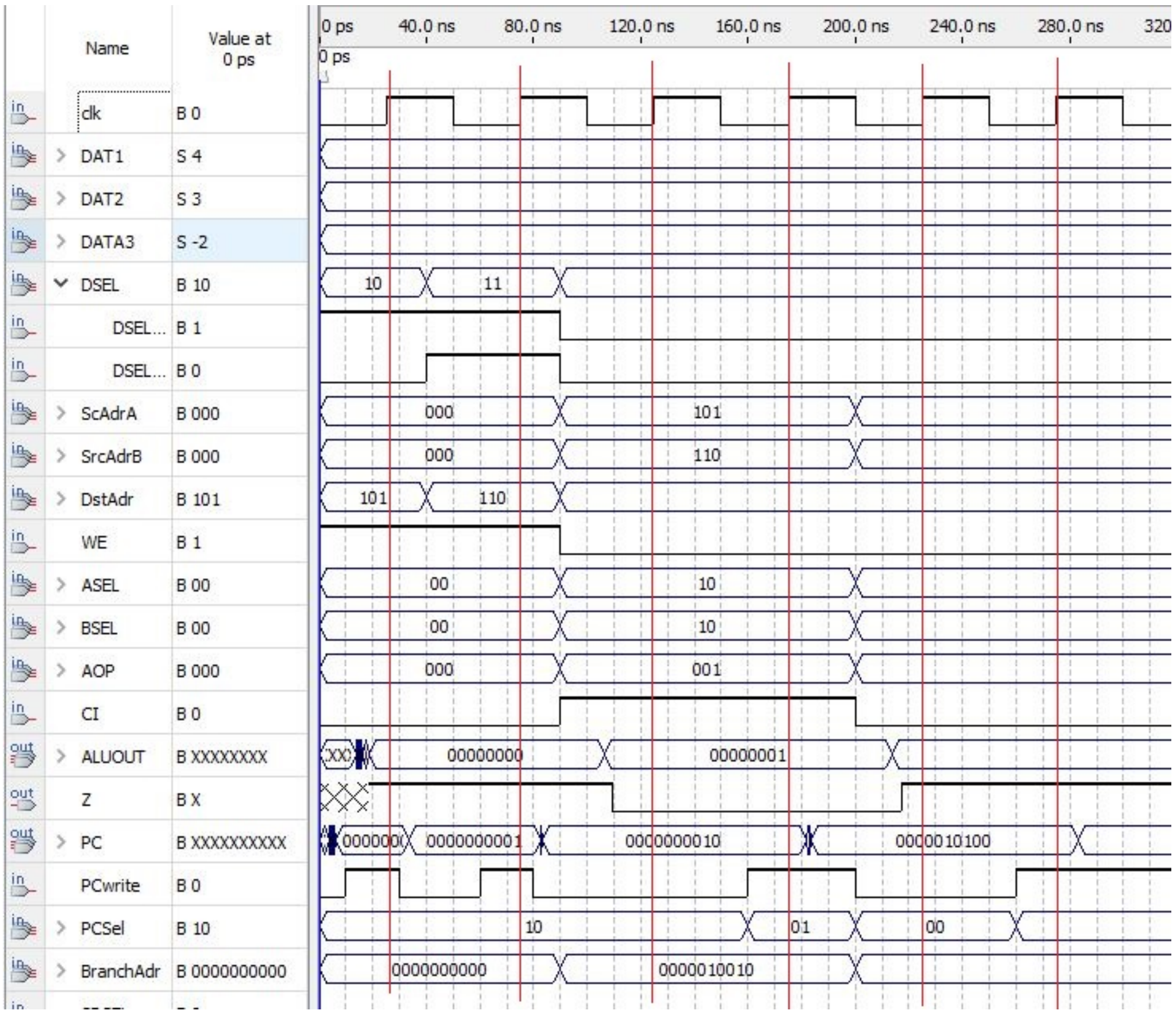
MOVI,ADDI,SUBBI



```
MOVI $5, 4
MOVI $6, 3
ADDI $6,$5,2
SUBI $5,$6,2
```

Lines added to track every cycle, MOVI takes 1 cycle while ADDI and SUBI takes 3 cycles. While PC increases 1 for each instruction.

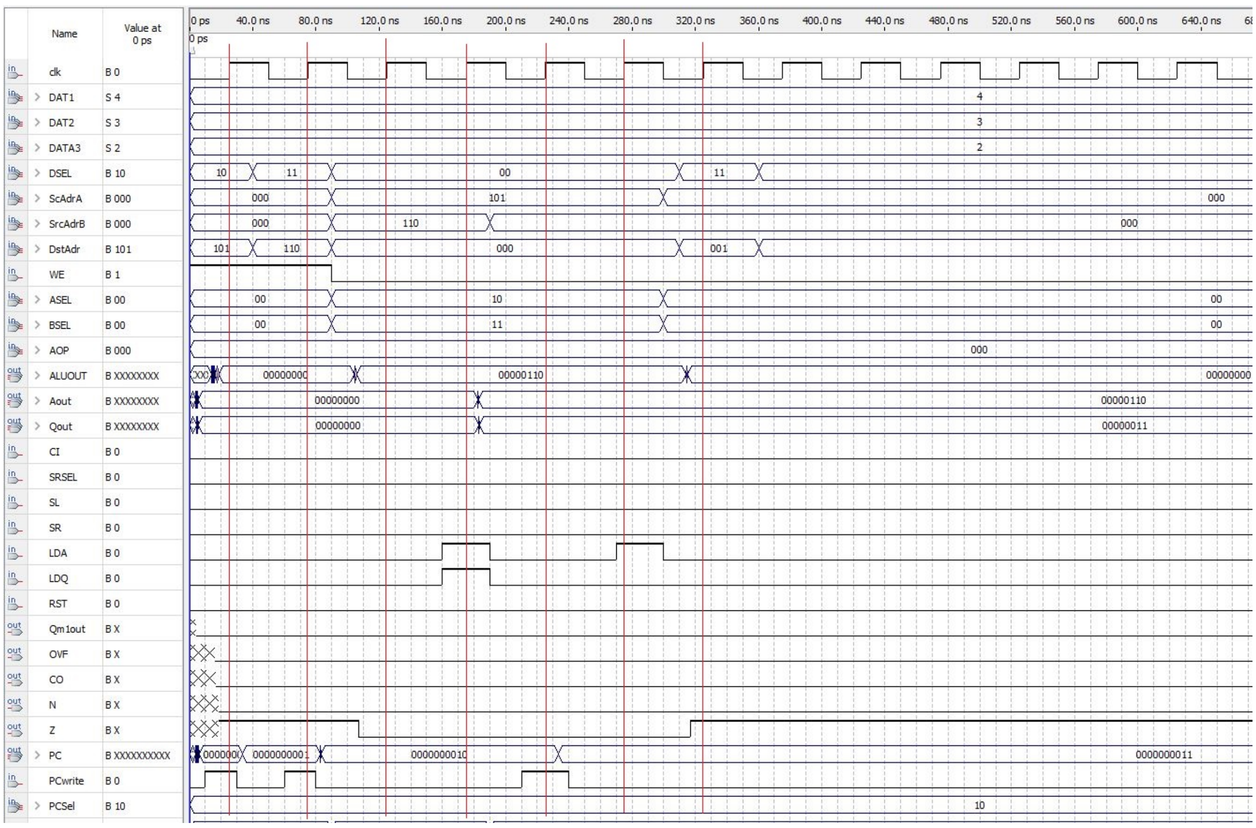
BEQ,BNEQ



```
MOVI $5, 4
MOVI $6, 3
BEQ $6,$5,Label
BNEQ $5,$6,Label
```

Lines added to track every cycle, MOVI takes 1 cycle while BEQ and BNEQ takes 2 cycles. In this simulation BEQ is not taken while BNEQ is taken. PC change can be seen here.

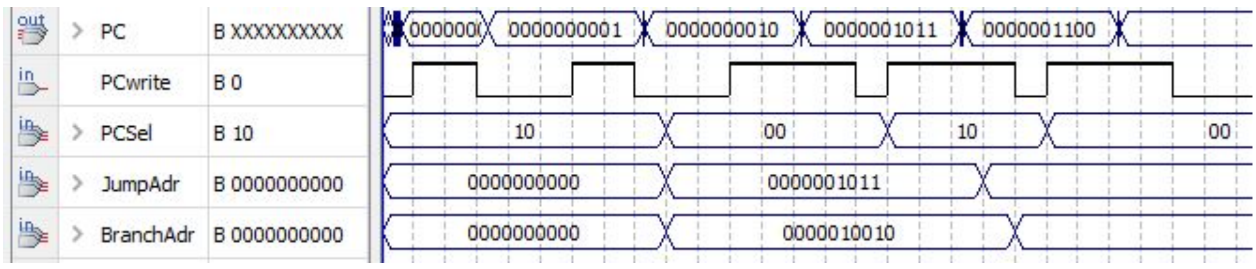
MOVI,LB,SB,LBU,SBU



```
MOVI $5, 4
MOVI $6, 3
SB $6,2($5)
LB $6,2($5)
```

Lines added to track every cycle, MOVI takes 1 cycle while SB takes 3 cycles while LD takes 4 cycles. SBU and LDU instructions have same control signal except mux signal after data memory output which selects high or low bits of 16-bit word. PC change can be seen here.

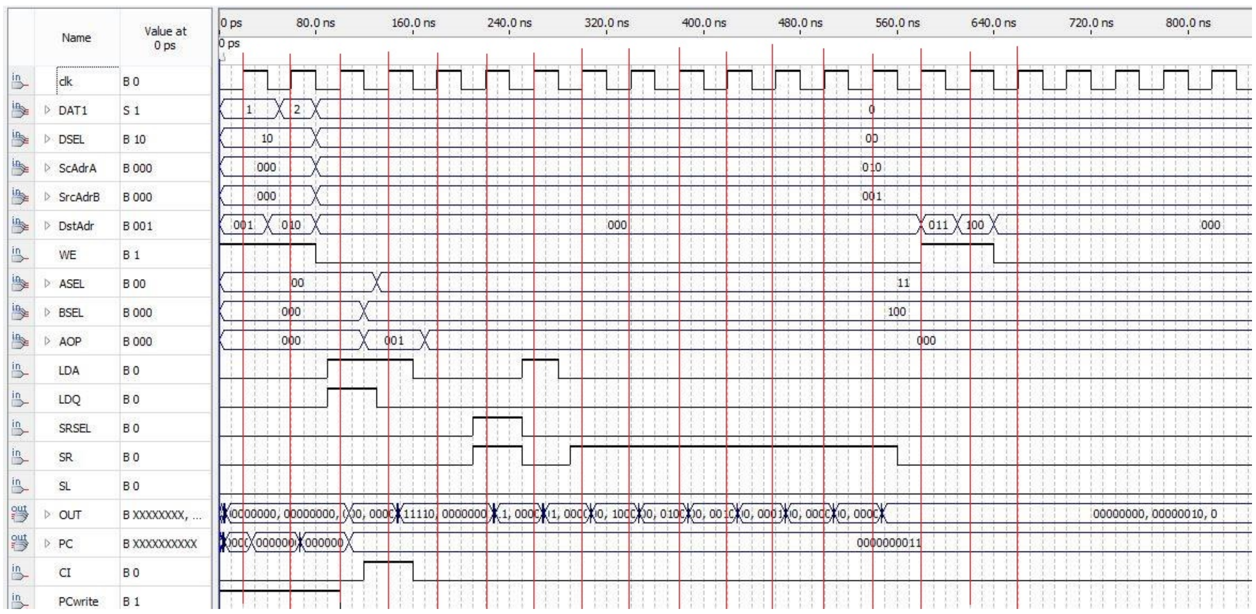
JMP



```
JMP Label
```

Jump directly replaces PC content.

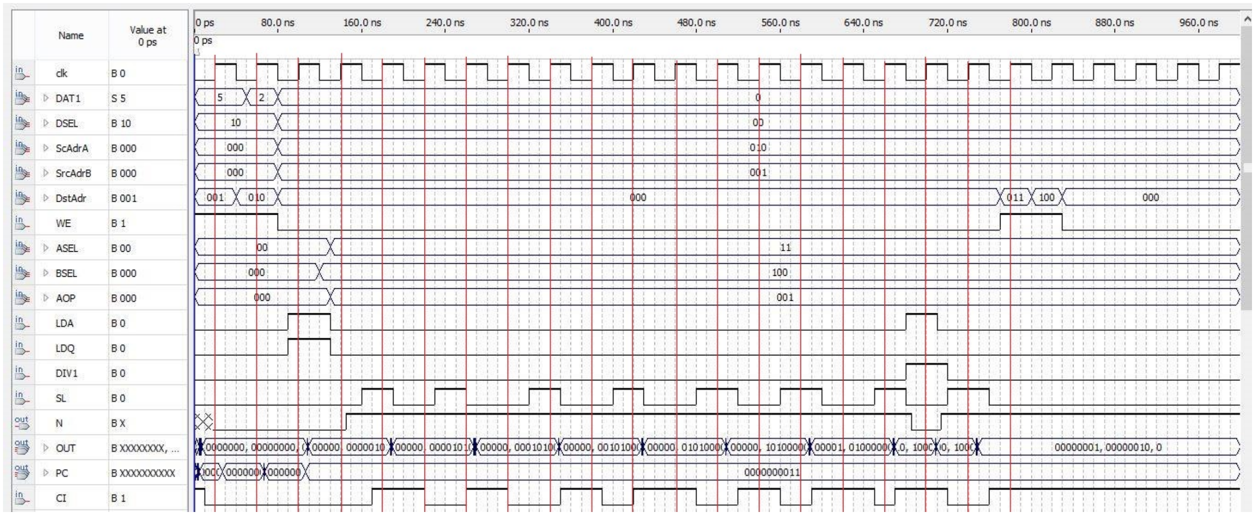
MOVI, MULT



```
MULT $3, $4, $1, $2
```

Lines added to track every cycle, MOVI takes 1 cycle while MULT cycle count changes depending on the number. Booth's algorithm used for MULT. It can take at most 20 cycles. It stores high byte to one register, while in lower in another byte that takes two cycles.

MOVI , DIV



```
DIV $3,$4,$1,$2
```

Lines added to track every cycle, MOVI takes 1 cycle while DIV cycle count changes depending on the number. It can take at most 20 cycles. It stores Quotient to one register, while remainder to another byte that takes two cycles.

Benchmark Programs and Test Results

Requested benchmarks are written in our ISA assembly code and run in our CPU design. All test results on memory can be seen.

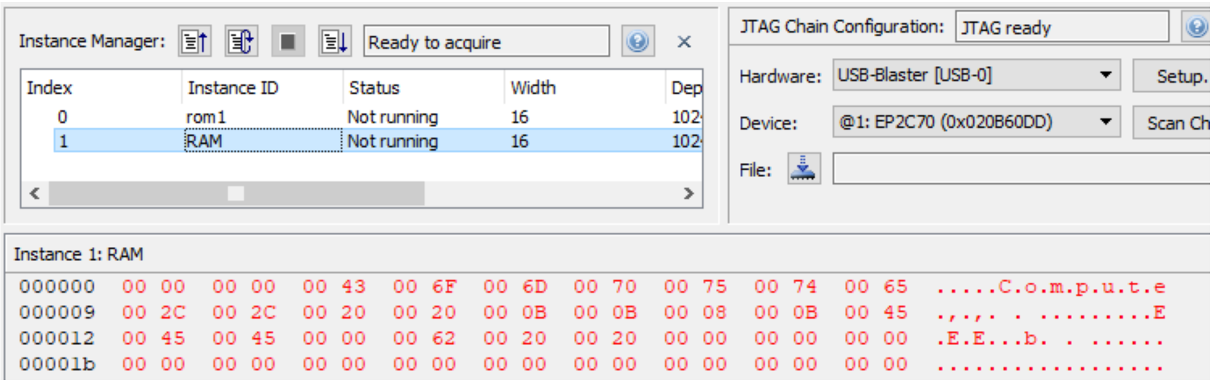
TEXT PARSER

```
movi $0,0

movi $1,-1

movi $2,1

countloop:movi $6,0

addi $2,$2,1

lb $3,0($2)

movi $5,8

movi $6,13

slt $3,$5,$7

beq $7,$1,notspace

slt $3,$6,$7

bneq $7,$1,notspace

addi $4,$4,1

jmp countloop

notspace:movi $6,32

bneq $3,$6,otherchar

addi $4,$4,1

jmp countloop

otherchar:movi $5,65

movi $6,90

slt $3,$5,$7

beq $7,$1,label2

slt $3,$6,$7

bneq $7,$1,label2

addi $0,$0,1

jmp countloop

label2:movi $5,97

movi $6,122

slt $3,$5,$7

beq $7,$1,label3

slt $3,$6,$7

bneq $7,$1,label3
```

```
addi $0,$0,1

jmp countloop

label3:movi $5,48

movi $6,57

slt $3,$5,$7

beq $7,$1,label4

slt $3,$6,$7

bneq $7,$1,label4

addi $0,$0,1

jmp countloop

label4:movi $6,0

beq $3,$6,end

jmp countloop

end:sb $0,0($6)

sb $4,1($6)

movi $0,0

movi $1,-1

movi $2,1

movi $3,2

loop:movi $6,0

addi $2,$2,1

lb $4,0($2)

beq $6,$4,extconsc

movi $5,8

movi $6,13

slt $4,$5,$7

beq $7,$1,nspc

slt $4,$6,$7

bneq $7,$1,nspc

oldvalcomp:beq $0,$4,loop

saveval:addi $0,$4,0

sbu $4,0($3)

addi $3,$3,1

jmp loop

nspc:movi $6,32

beq $4,$6,oldvalcomp

jmp saveval
```

```
extconsc:movi $3,1
movi $2,1
movi $6,0
rewrite:addi $3,$3,1
addi $2,$2,1
lbu $1,0($3)
sb $1,0($2)
sbu $6,0($3)
bneq $1,$6,rewrite
here:jmp here
```

SRAM content before the execution of code



Instance Manager: Ready to acquire

Index	Instance ID	Status	Width	Dep
0	rom1	Not running	16	102
1	RAM	Not running	16	102

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0]

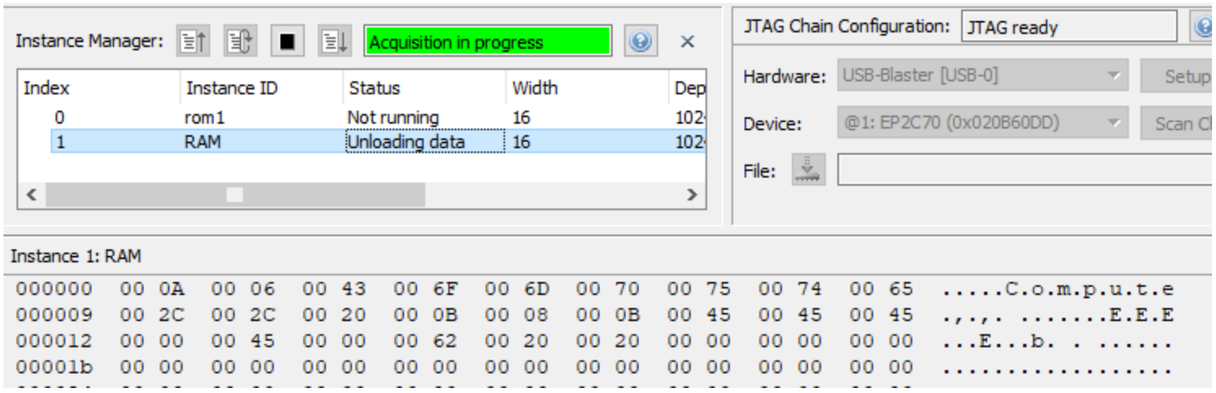
Device: @1: EP2C70 (0x020B60DD)

File:

Instance 1: RAM

```
000000  00 00  00 00  00 43  00 6F  00 6D  00 70  00 75  00 74  00 65  .....C.o.m.p.u.t.e
000009  00 2C  00 2C  00 20  00 20  00 0B  00 0B  00 08  00 0B  00 45  .,.,. . .........E
000012  00 45  00 45  00 00  00 62  00 20  00 20  00 00  00 00  00 00  .E.E...b. . ......
00001b  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  ..................
```

SRAM content after the execution of code



Instance Manager: Acquisition in progress

Index	Instance ID	Status	Width	Dep
0	rom1	Not running	16	102
1	RAM	Unloading data	16	102

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0]

Device: @1: EP2C70 (0x020B60DD)

File:

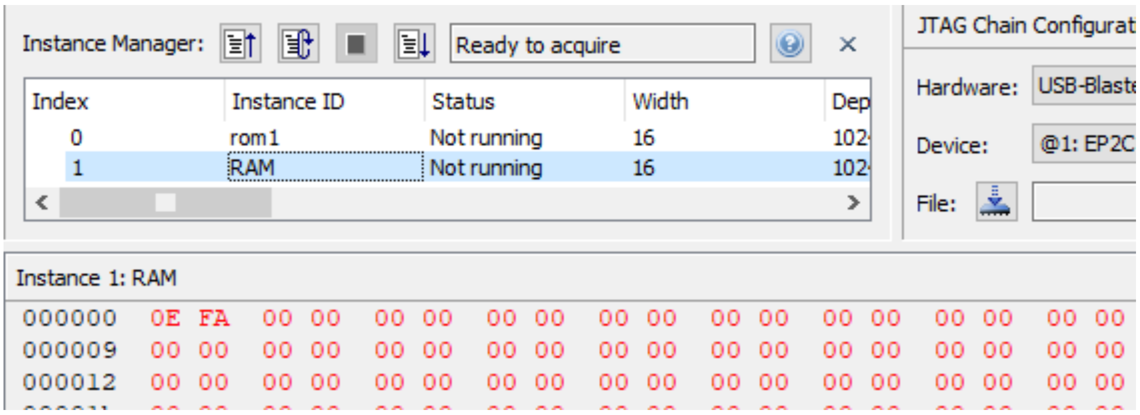
Instance 1: RAM

```
000000  00 0A  00 06  00 43  00 6F  00 6D  00 70  00 75  00 74  00 65  .....C.o.m.p.u.t.e
000009  00 2C  00 2C  00 20  00 0B  00 08  00 0B  00 45  00 45  00 45  .,.,. .......E.E.E
000012  00 00  00 45  00 00  00 62  00 20  00 20  00 00  00 00  00 00  ...E...b. . ......
00001b  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  ..................
```

MULTIPLICATION

```
movı $t0, 0

lb $t3, 0($t0)

lb $t4, 1($t0)

mult $t6, $t5, $t4, $t3

sb $t5, 2($t0)

sbu $t6, 0($t0)

here : jmp here
```

SRAM content before the execution



Instance Manager: Ready to acquire

JTAG Chain Configurat

Index	Instance ID	Status	Width	Dep
0	rom1	Not running	16	102
1	RAM	Not running	16	102

Hardware: USB-Blaste

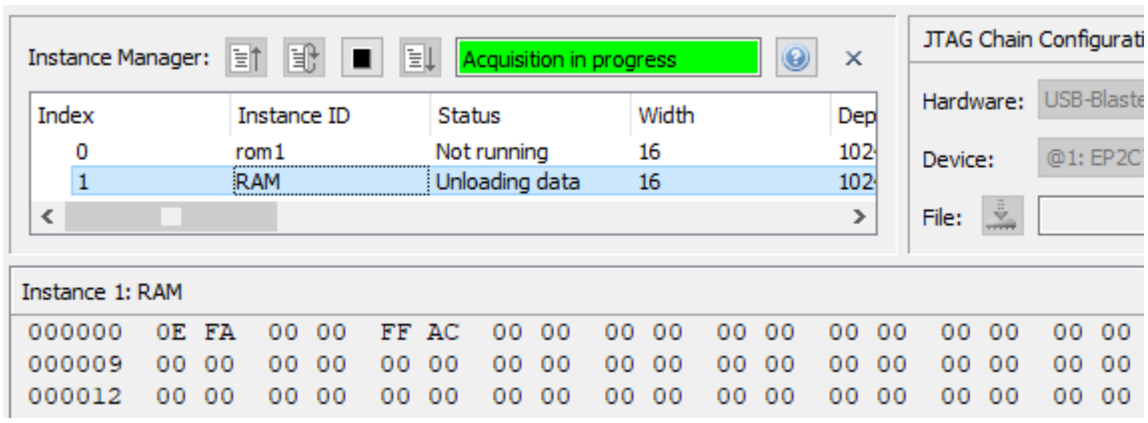
Device: @1: EP2C

File:

Instance 1: RAM

```
000000  0E FA  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00
000009  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00
000012  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00
```

SRAM content after the execution



Instance Manager: Acquisition in progress

JTAG Chain Configurati

Index	Instance ID	Status	Width	Dep
0	rom1	Not running	16	102
1	RAM	Unloading data	16	102

Hardware: USB-Blaste

Device: @1: EP2C

File:

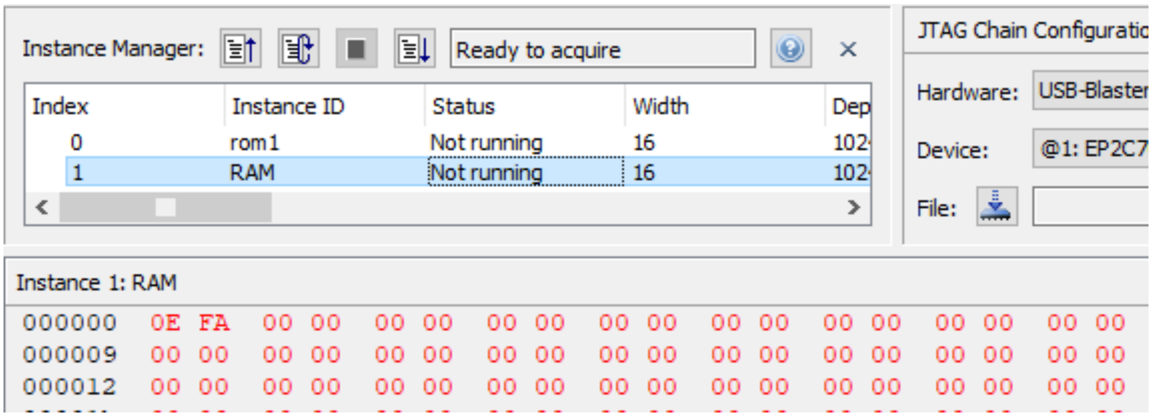
Instance 1: RAM

```
000000  0E FA  00 00  FF AC  00 00  00 00  00 00  00 00  00 00  00 00
000009  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00
000012  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00
```

DIVISION

```
movi $t0, 0

lb $t3, 0($t0)

lb $t4, 1($t0)

div $t6, $t5, $t4, $t3

sb $t5, 2($t0)

sbu $t6, 0($t0)

here : jmp here
```

SRAM content before the execution



Instance Manager: Ready to acquire

Index	Instance ID	Status	Width	Dep
0	rom1	Not running	16	102
1	RAM	Not running	16	102

JTAG Chain Configuratio

Hardware: USB-Blaster

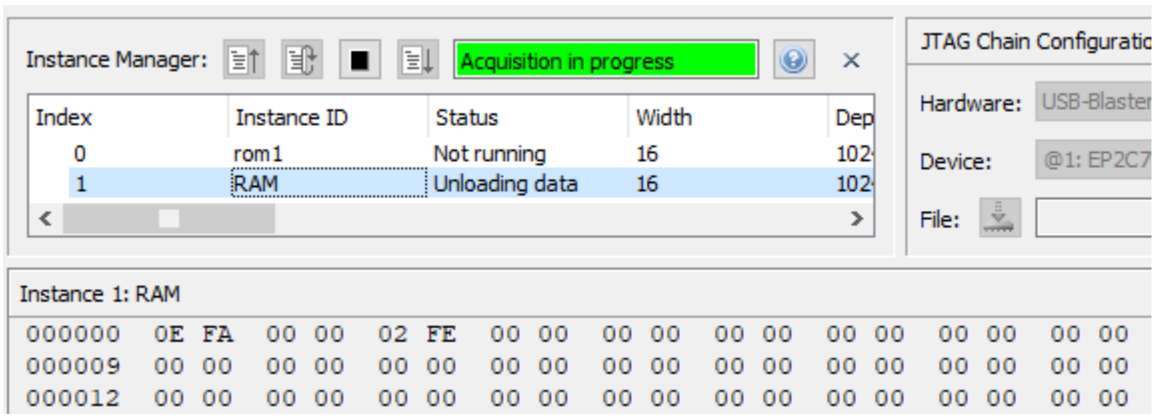
Device: @1: EP2C7

File:

Instance 1: RAM

```
000000  0E FA  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00
000009  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00
000012  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00
```

SRAM content after the execution



Instance Manager: Acquisition in progress

Index	Instance ID	Status	Width	Dep
0	rom1	Not running	16	102
1	RAM	Unloading data	16	102

JTAG Chain Configuratio

Hardware: USB-Blaster

Device: @1: EP2C7

File:

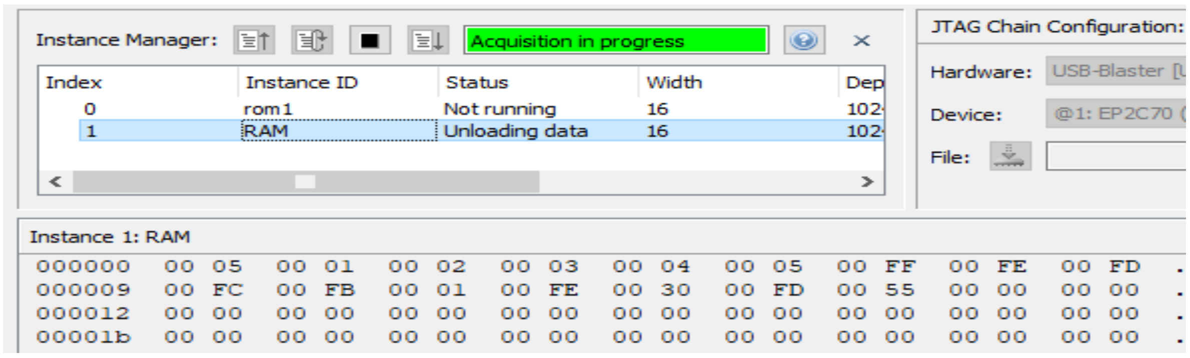
Instance 1: RAM

```
000000  0E FA  00 00  02 FE  00 00  00 00  00 00  00 00  00 00  00 00
000009  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00
000012  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00
```

N-LONG ARRAY COMPUTATION

```
movi $t0 , 0

movi $t7 , 0

lb $t1, 0($t0)

addi $t3, $t1,$t1

addi $t5, $t3, $t1

loop :

addi $t0,$t0,1

lb $t2 , 0($t0)

add $t7,$t7,$t2

addi $t4,$t1,$t0

lb $t2 , 0($t4)

add $t7,$t7,$t2

add $t6,$t3,$t0

lb $t2 , 0($t6)

sub $t7,$t7,$t2

add $t2,$t5, $t0

sb $t7, 0($t2)

bneq $t0 , $t1 , loop

here : jmp here
```

SRAM content before the execution

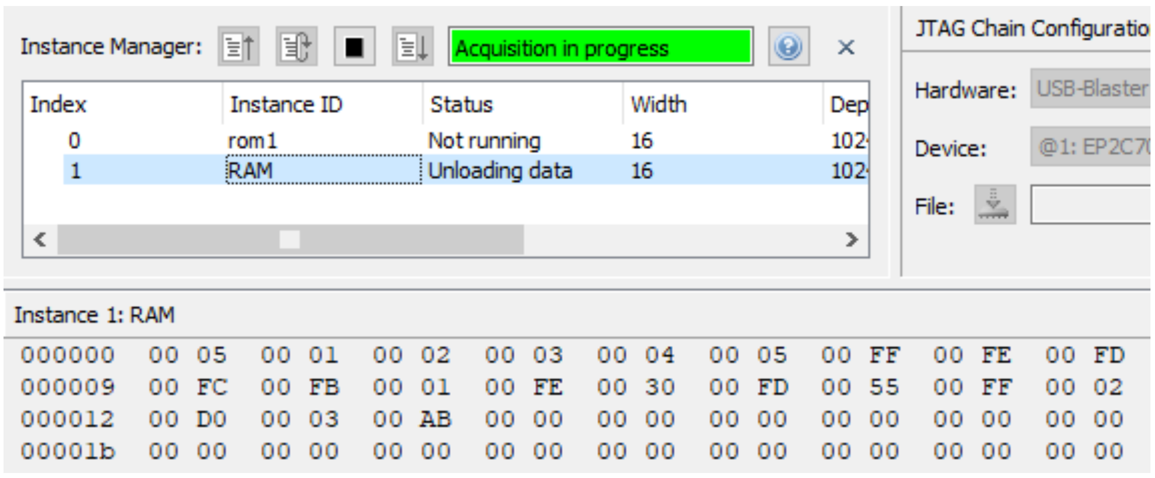


Index	Instance ID	Status	Width	Dep
0	rom1	Not running	16	102
1	RAM	Unloading data	16	102

Instance 1: RAM

```
000000   00 05   00 01   00 02   00 03   00 04   00 05   00 FF   00 FE   00 FD   .
000009   00 FC   00 FB   00 01   00 FE   00 30   00 FD   00 55   00 00   00 00   .
000012   00 00   00 00   00 00   00 00   00 00   00 00   00 00   00 00   00 00   .
00001b   00 00   00 00   00 00   00 00   00 00   00 00   00 00   00 00   00 00   .
```

SRAM content after the execution


Instance Manager: Acquisition in progress

Index	Instance ID	Status	Width	Dep
0	rom1	Not running	16	102
1	RAM	Unloading data	16	102

JTAG Chain Configuratio

Hardware: USB-Blaster

Device: @1: EP2C7

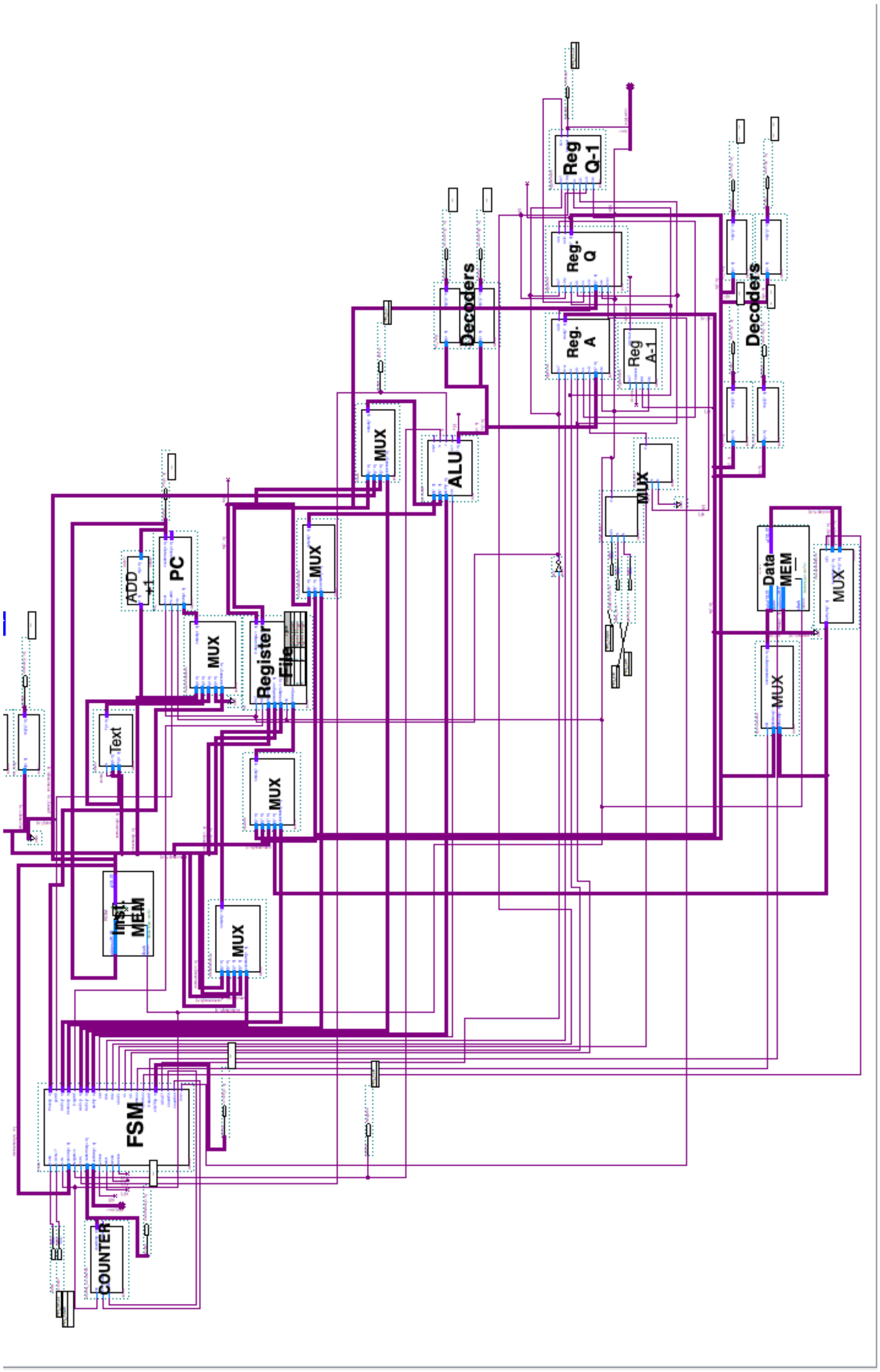
File:

Instance 1: RAM

```
000000   00 05   00 01   00 02   00 03   00 04   00 05   00 FF   00 FE   00 FD
000009   00 FC   00 FB   00 01   00 FE   00 30   00 FD   00 55   00 FF   00 02
000012   00 D0   00 03   00 AB   00 00   00 00   00 00   00 00   00 00   00 00
00001b   00 00   00 00   00 00   00 00   00 00   00 00   00 00   00 00   00 00
```

Design Files

Datapath Block Diagram



VHDL Codes

FSM

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity destinationmux is
       port(
              A, B , C, D                         : in std_logic_vector(2 downto 0);
              output                  : out std_logic_vector(2 downto 0);
              Selectbits              : in std_logic_vector(1 downto 0)
              );
end destinationmux;


architecture selector of destinationmux is
begin
              process(Selectbits)
              begin
                     case Selectbits is

                            when "00" => output <= A;

                            when "01" => output <= B;

                            when "10" => output <= C;

                            when "11" => output <= D;

                            when others =>

                     end case;
              end process;
end selector;


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity muxupdownByte is
    Port ( SEL : in  STD_LOGIC;
           DownByte   : in  STD_LOGIC_VECTOR(7 downto 0);
           UpByte     : in  STD_LOGIC_VECTOR(7 downto 0);
           OutByte   : out STD_LOGIC_VECTOR(7 downto 0));
end muxupdownByte ;

architecture Behavioral of muxupdownByte is
begin
   process(SEL)
              begin
                     case SEL is

                            when '0' => OutByte <= DownByte;

                            when '1' => OutByte <= UpByte;

                            when others =>
```

```vhdl
                      end case;
       end process;
end Behavioral;

LIBRARY ieee;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
USE ieee.numeric_std.ALL;

Entity FSM is
port(
      RUN : in std_logic;
          CLRINIT : in std_logic;
          CLK        : in std_logic;
              OpCode  : in std_logic_vector(3 downto 0) ;
          Negative: in std_logic ;
              Zero       : in std_logic ;
              CounterVal : in std_logic_VECTOR(3 downto 0);
              LastBits: in std_logic_VECTOR(1 downto 0);
              Qmsb           :in std_logic;
              Abef           :in std_logic;
              Amsb           :in std_logic;
              Mmsb           :in std_logic;
              PCsel   : out std_logic_vector(1 downto 0) ;
          pcWE    : out std_logic ;
          DSEL    : out std_logic_vector(2 downto 0);
              DestSel :out std_logic_vector(1 downto 0);
          RegWE   : out std_logic ;
          ASEL    : out std_logic_vector(1 downto 0);
      BSEL    : out std_logic_vector(2 downto 0);
              AOP     : out std_logic_vector(2 downto 0);
              Cin     : out std_logic ;
              LDA     : out std_logic ;
              LDQ     : out std_logic ;
              SRSEL   : out std_logic ;
              SL      : out std_logic ;
              SR      : out std_logic ;
              BytWrSel: out std_logic ;
              BytRrSel: out std_logic ;
              DataWE  : out std_logic ;
              STATE   : out std_logic_vector (5 downto 0);
              RESET   : out std_logic ;
              CountEn : out std_logic ;
              CountRst: out std_logic ;
          DIV1    : out std_logic                  );
end FSM ;

architecture behavior of FSM is

SIGNAL y_present, y_next : STD_LOGIC_VECTOR(36 DOWNTO 0);
CONSTANT s0  : STD_LOGIC_VECTOR(36 DOWNTO 0) := "1000000000000000000000000000000000000";
CONSTANT s1  : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0100000000000000000000000000000000000";
CONSTANT s2  : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0010000000000000000000000000000000000";
CONSTANT s3  : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0001000000000000000000000000000000000";
CONSTANT s4  : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000100000000000000000000000000000000";
CONSTANT s5  : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000010000000000000000000000000000000";
CONSTANT s6  : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000001000000000000000000000000000000";
CONSTANT s7  : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000100000000000000000000000000000";
CONSTANT s8  : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000010000000000000000000000000000";
CONSTANT s9  : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000001000000000000000000000000000";
CONSTANT s10 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000100000000000000000000000000";
CONSTANT s11 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000010000000000000000000000000";
CONSTANT s12 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000001000000000000000000000000";
CONSTANT s13 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000100000000000000000000000";
CONSTANT s14 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000010000000000000000000000";
CONSTANT s15 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000001000000000000000000000";
CONSTANT s16 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000100000000000000000000";
CONSTANT s17 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000010000000000000000000";
CONSTANT s18 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000001000000000000000000";
CONSTANT s19 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000100000000000000000";
CONSTANT s20 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000010000000000000000";
CONSTANT s21 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000001000000000000000";
CONSTANT s22 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000000100000000000000";
CONSTANT s23 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000000010000000000000";
CONSTANT s24 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000000001000000000000";
CONSTANT s25 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000000000100000000000";
CONSTANT s26 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000000000010000000000";
CONSTANT s27 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000000000001000000000";
CONSTANT s28 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000000000000100000000";
CONSTANT s29 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000000000000010000000";
CONSTANT s30 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000000000000001000000";
CONSTANT s31 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000000000000000100000";
CONSTANT s32 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000000000000000010000";
CONSTANT s33 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000000000000000001000";
CONSTANT s34 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000000000000000000100";
CONSTANT s35 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000000000000000000010";
CONSTANT s36 : STD_LOGIC_VECTOR(36 DOWNTO 0) := "0000000000000000000000000000000000001";
BEGIN

PROCESS (CLK,CLRINIT)
       BEGIN
              IF CLRINIT='1' THEN
                     y_present <= s0;
                     RESET <= '0';
              ELSIF (CLK'EVENT AND CLK='0' AND CLRINIT='0') THEN
                     y_present <= y_next;
                     RESET <= '1';
              END IF;

       END PROCESS;

PROCESS(OpCode,RUN,y_present,Zero)
     BEGIN
         CASE y_present IS

                      WHEN s0 =>
                      IF RUN='1' THEN
                      y_next <= s1 ;
                      ELSE
                      y_next <= s0 ;
                      END IF;

                      WHEN s1=>

                      IF OpCode ="1011" then
                      y_next <= s2 ;

                      ELSIF OpCode = "1010" then
                      y_next <= s4;

                      ELSIF OpCode = "1100" then
                      y_next <= s5;
```

```
ELSIF OpCode = "0000" then
y_next <= s6 ;

ELSIF OpCode = "0001" then
y_next <= s7 ;

ELSIF OpCode = "0110" then
y_next <= s9 ;

ELSIF OpCode = "0100" then
y_next <= s10 ;

ELSIF OpCode = "0101" then
y_next <= s11 ;

ELSIF OpCode = "0111" then
y_next <= s15 ;

ELSIF OpCode = "1111" then
y_next <= s15 ;

ELSIF OpCode = "1000" then
y_next <= s15 ;

ELSIF OpCode = "1110" then
y_next <= s15 ;

ELSIF OpCode = "1101" then
y_next <= s7 ;

ELSIF OpCode = "0010" then
y_next <= s22 ;

ELSIF OpCode = "0011" then
y_next <= s28 ;


ELSE
y_next <=s0 ;
END IF;

WHEN s2 =>
y_next <= s3;

WHEN s3 =>
y_next <= s0;

WHEN s4 =>
y_next <= s0 ;

WHEN s5 =>
y_next <= s3;


WHEN s6 =>
y_next <= s8 ;

WHEN s7 =>
IF Opcode="0001" then
      y_next <= s8;
ElSIF Opcode="1101" then
       IF Negative='0' then
       y_next <= s20 ;
       ELSE
       y_next <= s21;
       END IF;
END IF;

WHEN s8 =>
y_next <= s0 ;

WHEN s9 =>
y_next <= s0 ;

WHEN s10 =>
IF Zero='1' then
y_next <= s12 ;
ELSE
y_next <= s13;
END IF;

WHEN s11 =>
IF Zero='0'then
y_next <= s14 ;
ELSE
y_next <= s13;
END IF;

WHEN s12 =>
y_next<= s0;

WHEN s13 =>
y_next<= s0;

WHEN s14 =>
y_next<= s0;

WHEN s15 =>
IF Opcode="0111" then
y_next <= s16 ;
ELSIF Opcode="1111" then
y_next <= s17;
ELSIF Opcode="1000" then
y_next <= s18;
ELSIF Opcode="1110" then
y_next <= s19;
END IF;

WHEN s16 =>
y_next<= s0;
WHEN s17 =>
y_next<= s0;
WHEN s18 =>
y_next<= s0;
WHEN s19 =>
y_next<= s0;

WHEN s20 =>
y_next<= s0;
WHEN s21 =>
y_next<= s0;

WHEN s22 =>
IF CounterVal = "0000" THEN y_next <= s26;
```

```
              ELSE
                     IF LastBits = "10" THEN y_next <= s23;
                     ELSIF LastBits = "01" THEN y_next <= s24;
                     ELSE y_next <= s25;
                     END IF;
              END IF;

        WHEN s23 =>
        y_next<= s25;

        WHEN s24 =>
        y_next<= s25;

       WHEN s25 =>
              IF CounterVal = "0000" THEN y_next <= s26;
              ELSE
                     IF LastBits = "10" THEN y_next <= s23;
                     ELSIF LastBits = "01" THEN y_next <= s24;
                     ELSE y_next <= s25;
                     END IF;
              END IF;

        WHEN s26 =>
        y_next<= s27;

        WHEN s27 =>
        y_next<= s0;

        WHEN s28 =>
        IF Qmsb = '1' then
        y_next<= s29;
        else
        y_next<=s30;
        end if;

        WHEN s29 =>
        y_next<= s30;


        WHEN s30 =>
    IF Abef = Mmsb then
              y_next <= s32;
        Else
          y_next <= s31;
        end if;


        WHEN s31 =>
        IF (Zero = '1' or Amsb = Abef) then
              y_next <= s33;
        elsif counterVal = "0000" then
         y_next <= s35;
        else
              y_next <= s30;
        end if;

        WHEN s32 =>
        IF Zero = '1' then
              y_next <= s34;
       elsif (Amsb = Abef) then
              y_next <= s34;
```

```
                        elsif counterVal = "0000" then
                         y_next <= s35;
                        else
                              y_next <= s30;
                        end if;

                        WHEN s33 =>
                         if counterVal = "0000" then
                                     y_next <= s35 ;
                              else
                                     y_next <= s30;
                              end if;

                       WHEN s34 =>
                         if counterVal = "0000" then
                                     y_next <= s35 ;
                              else
                                     y_next <= s30;
                              end if;

                        WHEN s35 =>
                        y_next <= s36 ;

                        WHEN s36 =>
                        y_next <= s0 ;

                        WHEN others =>
                        y_next <= s0;

          END CASE;
          END PROCESS;


PROCESS(y_present)
     BEGIN

          IF y_present = s0 then
               pcWE <= '0';
           PCsel <= "00";
           DSEL    <= "000";
               DestSel <= "00" ;
           RegWE   <= '0' ;
           ASEL    <= "00" ;
      BSEL    <= "000" ;
               AOP     <= "000" ;
               Cin     <= '0' ;
               LDA     <= '0' ;
               LDQ     <= '0' ;
               SRSEL   <= '0' ;
               SL      <= '0' ;
               SR      <= '0' ;
               BytWrSel<= '0' ;
               BytRrSel<= '0' ;
           DataWE  <= '0' ;
               CountEn <= '0' ;
               CountRst<= '0' ;
               DIV1    <= '0' ;
       STATE   <= "000000" ;
               END IF ;

               IF y_present = s1 then
               pcWE <= '0';
```

```
        PCsel <= "00";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "00" ;
BSEL    <= "000" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
    DataWE  <= '0' ;
        CountEn <= '0' ;
        DIV1    <= '0' ;
        STATE   <="000001";
        END IF;

        IF y_present = s2 then
        pcWE <= '0';
        PCsel <= "10";
    DSEL    <= "001";
        DestSel <= "01" ;
    RegWE   <= '0' ;
    ASEL    <= "10" ;
BSEL    <= "011" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '1' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
    DataWE  <= '0' ;
        DIV1    <= '0' ;
        STATE   <="000010";
        END IF ;

        IF y_present = s3 then
        pcWE <= '1';
        PCsel <= "10";
    DSEL    <= "001";
        DestSel <= "01" ;
    RegWE   <= '1' ;
    ASEL    <= "00" ;
BSEL    <= "000" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
    DataWE  <= '0' ;
        DIV1    <= '0' ;
        STATE   <= "000011";
        END IF ;

        IF y_present = s4 then
        pcWE <= '1';
        PCsel <= "10";
    DSEL    <= "010";
        DestSel <= "00" ;
    RegWE   <= '1' ;
    ASEL    <= "00" ;
BSEL    <= "000" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
    DataWE  <= '0' ;
        DIV1    <= '0' ;
        STATE   <="000100";
end if ;

        IF y_present = s5 then
    pcWE <= '0';
        PCsel <= "10";
    DSEL    <= "001";
        DestSel <= "01" ;
    RegWE   <= '0' ;
    ASEL    <= "10" ;
BSEL    <= "011" ;
        AOP     <= "001" ;
        Cin     <= '1' ;
        LDA     <= '1' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
    DataWE  <= '0' ;
        DIV1    <= '0' ;
        STATE   <="000101";
        END IF ;

        IF y_present = s6 then

    pcWE <= '0';
        PCsel <= "00";
    DSEL    <= "000";
        DestSel <= "10" ;
    RegWE   <= '0' ;
    ASEL    <= "10" ;
BSEL    <= "010" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '1' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
    DataWE  <= '0' ;
        DIV1    <= '0' ;
        STATE   <="000110";
        END IF ;

        IF y_present = s7 then

    pcWE <= '0';
        PCsel <= "00";
    DSEL    <= "000";
        DestSel <= "10" ;
    RegWE   <= '0' ;
    ASEL    <= "10" ;
BSEL    <= "010" ;
        AOP     <= "001" ;
        Cin     <= '1' ;
        LDA     <= '1' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
    DataWE  <= '0' ;
        DIV1    <= '0' ;
        STATE   <="000111";
        END IF ;

        IF y_present = s8 then

    pcWE <= '1';
        PCsel <= "10";
    DSEL    <= "001";
        DestSel <= "10" ;
    RegWE   <= '1' ;
    ASEL    <= "00" ;
BSEL    <= "000" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
    DataWE  <= '0' ;
        DIV1    <= '0' ;
        STATE   <="001000";
        END IF ;

        IF y_present = s9 then

    pcWE <= '1';
        PCsel <= "00";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "00" ;
```

```
BSEL    <= "000" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
    DataWE  <= '0' ;
        DIV1    <= '0' ;
        STATE   <="001001";
END IF ;

        IF y_present = s10 then

    pcWE <= '0';
        PCsel <= "00";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "10" ;
BSEL    <= "010" ;
        AOP     <= "001" ;
        Cin     <= '1' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
    DataWE  <= '0' ;
        DIV1    <= '0' ;
        STATE   <="001010";
END IF ;
        IF y_present = s11 then

    pcWE <= '0';
        PCsel <= "00";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "10" ;
BSEL    <= "010" ;
        AOP     <= "001" ;
        Cin     <= '1' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
    DataWE  <= '0' ;
        DIV1    <= '0' ;
        STATE   <="001011";
END IF ;

        IF y_present = s12 then
    pcWE <= '1';
        PCsel <= "01";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "00" ;
BSEL    <= "000" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
    DataWE  <= '0' ;
        DIV1    <= '0' ;
        STATE   <="001100";
END IF ;

        IF y_present = s13 then

    pcWE <= '1';
        PCsel <= "10";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "00" ;
BSEL    <= "000" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
    DataWE  <= '0' ;
        DIV1    <= '0' ;
        STATE   <="001101";
END IF ;

        IF y_present = s14 then

    pcWE <= '1';
        PCsel <= "01";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "00" ;
BSEL    <= "000" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
    DataWE  <= '0' ;
        DIV1    <= '0' ;
        STATE   <="001110";
END IF ;

        IF y_present = s15 then

    pcWE <= '0';
        PCsel <= "00";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "10" ;
BSEL    <= "011" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '1' ;
        LDQ     <= '1' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
    DataWE  <= '0' ;
        DIV1    <= '0' ;
        STATE   <="001111";
END IF ;

        IF y_present = s16 then

    pcWE <= '1';
        PCsel <= "10";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "10" ;
BSEL    <= "011" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '1' ;
        DataWE  <= '1' ;
    DIV1    <= '0' ;
        STATE   <="010000";
END IF ;

        IF y_present = s17 then

    pcWE <= '1';
        PCsel <= "10";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "10" ;
BSEL    <= "011" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '1' ;
        BytRrSel<= '0' ;
        DataWE  <= '1' ;
        DIV1    <= '0' ;

        STATE   <="010001";
END IF ;

        IF y_present = s18 then

    pcWE <= '1';
        PCsel <= "10";
    DSEL    <= "011";
        DestSel <= "01" ;
    RegWE   <= '1' ;
    ASEL    <= "10" ;
BSEL    <= "011" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
        DataWE  <= '0' ;
        DIV1    <= '0' ;

        STATE   <="010010";
END IF ;

        IF y_present = s19 then

    pcWE <= '1';
        PCsel <= "10";
    DSEL    <= "011";
        DestSel <= "01" ;
    RegWE   <= '1' ;
    ASEL    <= "10" ;
BSEL    <= "011" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '1' ;
        DataWE  <= '0' ;
        DIV1    <= '0' ;

        STATE   <="010011";
END IF ;

        IF y_present = s20 then
    pcWE <= '1';
        PCsel <= "10";
    DSEL    <= "101";
        DestSel <= "10" ;
    RegWE   <= '1' ;
    ASEL    <= "10" ;
BSEL    <= "011" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
        DataWE  <= '0' ;
        DIV1    <= '0' ;

        STATE   <="010100";
END IF ;

        IF y_present = s21 then

    pcWE <= '1';
        PCsel <= "10";
    DSEL    <= "100";
        DestSel <= "10" ;
    RegWE   <= '1' ;
    ASEL    <= "10" ;
BSEL    <= "011" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
        DataWE  <= '0' ;
        DIV1    <= '0' ;

        STATE   <="010101";
END IF ;

        IF y_present = s22 then

    pcWE <= '0';
        PCsel <= "10";
    DSEL    <= "100";
        DestSel <= "10" ;
    RegWE   <= '0' ;
    ASEL    <= "10" ;
BSEL    <= "000" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '1' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
        DataWE  <= '0' ;
        CountRst<= '1' ;
        CountEn <= '0' ;
        DIV1    <= '0' ;
        STATE   <="010110";
END IF ;

        IF y_present = s23 then

    pcWE <= '0';
        PCsel <= "10";
    DSEL    <= "100";
        DestSel <= "10" ;
    RegWE   <= '0' ;
    ASEL    <= "11" ;
BSEL    <= "100" ;
        AOP     <= "001" ;
        Cin     <= '1' ;
        LDA     <= '1' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
        DataWE  <= '0' ;
        CountRst<= '1' ;
    CountEn <= '0' ;
        DIV1    <= '0' ;
        STATE   <="010111";
END IF ;

        IF y_present = s24 then

    pcWE <= '0';
        PCsel <= "10";
    DSEL    <= "100";
        DestSel <= "10" ;
    RegWE   <= '0' ;
    ASEL    <= "11" ;
BSEL    <= "100" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '1' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
        DataWE  <= '0' ;
        CountRst<= '1' ;
    CountEn <= '0' ;
        DIV1    <= '0' ;
        STATE   <="011000";
END IF ;

        IF y_present = s25 then
    pcWE <= '0';
        PCsel <= "10";
    DSEL    <= "100";
        DestSel <= "10" ;
    RegWE   <= '0' ;
    ASEL    <= "10" ;
BSEL    <= "000" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '1' ;
        SL      <= '0' ;
        SR      <= '1' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
        DataWE  <= '0' ;
        CountEn <= '1' ;
        CountRst<= '1' ;
        DIV1    <= '0' ;

        STATE   <="011001";
END IF ;

        IF y_present = s26 then

    pcWE <= '0';
        PCsel <= "10";
    DSEL    <= "001";
        DestSel <= "10" ;
    RegWE   <= '1' ;
    ASEL    <= "10" ;
BSEL    <= "000" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
        DataWE  <= '0' ;
        CountRst<= '1' ;
    CountEn <= '0' ;
        DIV1    <= '0' ;
        STATE   <="011010";
END IF ;

        IF y_present = s27 then

    pcWE <= '1';
        PCsel <= "10";
    DSEL    <= "000";
        DestSel <= "11" ;
    RegWE   <= '1' ;
    ASEL    <= "10" ;
BSEL    <= "000" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
        DataWE  <= '0' ;
        CountRst<= '1' ;
        CountEn <= '0' ;
        DIV1    <= '0' ;

        STATE   <="011011";
END IF ;

        IF y_present = s28 then

    pcWE <= '0';
        PCsel <= "10";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "00" ;
BSEL    <= "000" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '1' ;
        LDQ     <= '1' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
        DataWE  <= '0' ;
        CountRst<= '0' ;
        CountEn <= '0' ;
        DIV1    <= '0' ;

        STATE   <="011100";
END IF ;

        IF y_present = s29 then

    pcWE <= '0';
        PCsel <= "10";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "01" ;
BSEL    <= "000" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '1' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
        DataWE  <= '0' ;
        CountRst<= '1' ;
        CountEn <= '0' ;
        DIV1    <= '0' ;
        STATE   <="011101";
END IF ;

        IF y_present = s30 then

    pcWE <= '0';
        PCsel <= "10";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "00" ;
BSEL    <= "000" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '1' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
        DataWE  <= '0' ;
        CountRst<= '1' ;
        CountEn <= '1' ;
        DIV1    <= '0' ;

        STATE   <="011110";
END IF ;

        IF y_present = s31 then

    pcWE <= '0';
        PCsel <= "10";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "11" ;
BSEL    <= "100" ;
        AOP     <= "000" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
        DataWE  <= '0' ;
        CountRst<= '1' ;
        CountEn <= '0' ;
        DIV1    <= '0' ;

        STATE   <="011111";
END IF ;

        IF y_present = s32 then

    pcWE <= '0';
        PCsel <= "10";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "11" ;
BSEL    <= "100" ;
        AOP     <= "001" ;
        Cin     <= '0' ;
        LDA     <= '0' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
        DataWE  <= '0' ;
        CountRst<= '1' ;
        CountEn <= '0' ;
        DIV1    <= '0' ;

        STATE   <="100000";
END IF ;

        IF y_present = s33 then

    pcWE <= '0';
        PCsel <= "10";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "11" ;
BSEL    <= "100" ;
        AOP     <= "000" ;
        Cin     <= '1' ;
        LDA     <= '1' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
        DataWE  <= '0' ;
        CountRst<= '1' ;
        CountEn <= '0' ;
        DIV1    <= '1' ;

        STATE   <="100001";
END IF ;
        IF y_present = s34 then

    pcWE <= '0';
        PCsel <= "10";
    DSEL    <= "000";
        DestSel <= "00" ;
    RegWE   <= '0' ;
    ASEL    <= "11" ;
BSEL    <= "100" ;
        AOP     <= "001" ;
        Cin     <= '1' ;
        LDA     <= '1' ;
        LDQ     <= '0' ;
        SRSEL   <= '0' ;
        SL      <= '0' ;
        SR      <= '0' ;
        BytWrSel<= '0' ;
        BytRrSel<= '0' ;
```

```
            DataWE  <= '0' ;
            CountRst<= '1' ;
            CountEn <= '0' ;
            DIV1    <= '1' ;

            STATE   <="100010";
    END IF ;

            IF y_present = s35 then

        pcWE <= '0';
            PCsel <= "10";
        DSEL    <= "001";
            DestSel <= "10" ;
        RegWE   <= '1' ;
        ASEL    <= "00" ;
    BSEL    <= "000" ;
            AOP     <= "000" ;
            Cin     <= '0' ;
            LDA     <= '0' ;
            LDQ     <= '0' ;
            SRSEL   <= '0' ;
            SL      <= '0' ;
            SR      <= '0' ;
            BytWrSel<= '0' ;
            BytRrSel<= '0' ;
            DataWE  <= '0' ;
            CountRst<= '0' ;
            CountEn <= '0' ;
            DIV1    <= '0' ;

            STATE   <="100011";
    END IF ;

            IF y_present = s36 then

        pcWE <= '1';
            PCsel <= "10";
        DSEL    <= "000";
            DestSel <= "11" ;
        RegWE   <= '1' ;
        ASEL    <= "00" ;
    BSEL    <= "000" ;
            AOP     <= "000" ;
            Cin     <= '0' ;
            LDA     <= '0' ;
            LDQ     <= '0' ;
            SRSEL   <= '0' ;
            SL      <= '0' ;
            SR      <= '0' ;
            BytWrSel<= '0' ;
            BytRrSel<= '0' ;
            DataWE  <= '0' ;
            CountRst<= '0' ;
            CountEn <= '0' ;
            DIV1    <= '0' ;

            STATE   <="100100";
    END IF ;
    END PROCESS ;
```

PC ADDER

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity Adder is
       port(
--             inputs:
               A              : in STD_LOGIC;
               B              : in std_LOGIC_VECTOR(5 downto 0);
               PC             : in std_LOGIC_VECTOR(9 downto 0);          -- operands
               F                      : out STD_LOGIC_VECTOR(9 downto 0)          -- result
       );
end Adder;

architecture calculation of Adder is
signal G : std_LOGIC_VECTOR(9 downto 0);
       begin
       process(A,B)
       Begin
                             G(9) <= A;
                             G(8) <= A;
                             G(7) <= A;
                             G(6) <= A;
                             G(5 downto 0) <= B;
       end process;
               F <= PC + G;
end architecture;
```

ALU (ARITHMETIC LOGIC UNIT)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity ALUunit2 is
       port(
--             inputs:
               A, B           : in signed(7 downto 0);            -- operands
               AOP            : in std_logic_vector(2 downto 0);  -- operation
               cin            : in std_logic;                             -- carry in,
clock, CE-input (whether component should be active or not)
--             outputs:
               cout, n, z, ovf: out std_logic;                    -- carry, sign, zero flag
               F              : out signed(7 downto 0)            -- result
       );
end ALUunit2;


architecture calculation of ALUunit2 is

       signal F_i : signed(8 downto 0) := "000000000";
       -- internal signal for calculation
       -- is assigned to F-output and carry-flag with concurrent statement

       begin

--     councurrent statements
```

```
       n <= F_i(7);
       -- sign-flag is determined by bit 15 of the result -> sign bit

       z <= '1' when F_i(7 downto 0) = "000000000" else '0';
       -- only setting zero flag if result is zero, so all bits of F have to be 0

       F    <= F_i(7 downto 0);
       -- bits 15 downto 0 will be the result

       cout <= F_i(8);
       -- bit 16 of F_i is the carry-flag

       ovf <= '1' when (AOP = "000" and A(7) = B(7) and A(7)/=F_i(7)) or (AOP = "001" and
A(7) /= B(7) and A(7) /= F_i(7)) or (AOP = "010" and A(7) /= B(7) and B(7) /= F_i(7)) else
'0';
--     processes

       process(AOP) is
       begin

                     case AOP is
                            -- determining operation
                            -- concatenating first when using arithmetic calculations
                            -- when using logical operations, the carry-flag is always 0

                            when "000" =>                              -- ADD
                                   F_i <= ('0' & A) + ('0' & B);

                            when "010" =>                              -- SUBR
                                     F_i <= ('0' & B) - ('0' & A) ;
                            when "001" =>                              -- SUBS
                                     F_i <= ('0' & A) - ('0' & B);


                            -- concatenation happening after calculation because carry
flag is impossible to reach
                            when "011" => F_i <= '0' & (A OR B);       -- OR
                            when "100" => F_i <= '0' & (A AND B);             -- AND
                            when "101" => F_i <= '0' & (NOT A AND B);  -- NOTRS
                            when "110" => F_i <= '0' & (A XOR B);             -- XOR
                            when "111" => F_i <= '0' & (A XNOR B);            -- XNOR
                            when others =>
                     end case;

       end process;
end architecture;
```

MUX 2 TO 1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux_2to1 is
    Port ( SEL : in  STD_LOGIC;
           A   : in  STD_LOGIC;
           B   : in  STD_LOGIC;
           X   : out STD_LOGIC);
end mux_2to1 ;

architecture Behavioral of mux_2to1 is
begin
```

```
    X <= A when (SEL = '1') else B;
end Behavioral;
```

MUX 4 TO 1

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity mux4to1 is
        port(
                A, B, C, D              : in std_logic_vector(7 downto 0);
                output                  : out std_logic_vector(7 downto 0);
                Selectbits              : in std_logic_vector(2 downto 0)
                );
end mux4to1;


architecture selector of mux4to1 is
begin
                process(Selectbits)
                begin
                        case Selectbits is

                                when "000" => output <= A;

                                when "001" => output <= B;

                                when "010" => output <= C;

                                when "011" => output <= D;

                                when "100" => output <= "11111111";

                                when "101" => output <= "00000000";

                                when others =>

                        end case;
                end process;
end selector;
```

MUX 4 TO 1 WITH ZERO AND 1 OPTION

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity mux4withzeroone is
        port(
                A, B                            : in std_logic_vector(7 downto 0);
                output                  : out std_logic_vector(7 downto 0);
                Selectbits              : in std_logic_vector(1 downto 0)
                );
end mux4withzeroone;


architecture selector of mux4withzeroone is
```

```
begin
              process(Selectbits)
              begin
                     case Selectbits is

                            when "00" => output <= "00000000";

                            when "01" => output <= "11111111";

                            when "10" => output <= A;

                            when "11" => output <= B;

                            when others =>

                     end case;
              end process;
end selector;
```

PROGRAM COUNTER (PC)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PC is

Port ( reset,PCwrite,clk : in STD_LOGIC;
          data : in STD_LOGIC_VECTOR (9 downto 0);
          counterout,pcplusone : out STD_LOGIC_VECTOR ( 9 downto 0 )
        );

       end PC ;

architecture behaviour of PC is
signal values : STD_LOGIC_VECTOR(9 downto 0);
begin
       counterout <= values;
       pcplusone <= values + 2 ;
       process(clk, reset, data,PCwrite)
       begin
              if reset = '1' then
                     values <= "0000000000";

              elsif clk'event and clk='1' then
                     if PCwrite ='1' then
                     values <= data;

              end if;
              end if;

              end process;
end behaviour;
```

REGISTER A

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

ENTITY registerA IS
PORT ( RST,LDA     : IN STD_LOGIC ;
       SL , SR     : IN STD_LOGIC ;
               SIL , SIR   : IN STD_LOGIC ;
               SOR         : OUT STD_LOGIC ;
       D           : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
               Clk         : IN STD_LOGIC;
               Aout        : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END registerA ;

ARCHITECTURE behavior OF registerA IS
SIGNAL Atry : STD_LOGIC_VECTOR(7 Downto 0);


BEGIN
                      SOR     <= Atry(0);
                Aout(7) <= Atry(7);
                Aout(6) <= Atry(6);
                Aout(5) <= Atry(5);
                Aout(4) <= Atry(4);
                Aout(3) <= Atry(3);
                Aout(2) <= Atry(2);
                Aout(1) <= Atry(1);
                Aout(0) <= Atry(0);

  PROCESS(Clk,RST,LDA)
  BEGIN

  IF(RST='0') Then

         IF(Clk 'EVENT and Clk='1') Then
            IF(LDA='1') Then
                Atry <= D ;

                ELSIF(SL='1' AND LDA='0') Then

                Atry(7) <= Atry(6);
                Atry(6) <= Atry(5);
                Atry(5) <= Atry(4);
                Atry(4) <= Atry(3);
                Atry(3) <= Atry(2);
                Atry(2) <= Atry(1);
                Atry(1) <= Atry(0);
                Atry(0) <= SIL ;

               ELSIF(SR='1' AND LDA='0' AND SL='0') Then
                Atry(7) <= SIR;
                Atry(6) <= Atry(7);
                Atry(5) <= Atry(6);
                Atry(4) <= Atry(5);
                Atry(3) <= Atry(4);
                Atry(2) <= Atry(3);
                Atry(1) <= Atry(2);
                Atry(0) <= Atry(1);
                END IF;
         END IF ;
  ELSIF(RST='1') Then
  Atry <= "00000000" ;
  END IF ;
  END PROCESS ;
END behavior ;
```

REGISTER M

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

ENTITY registerM IS
PORT ( RST,LDM  : IN STD_LOGIC ;
       D           : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
               Clk         : IN STD_LOGIC;
               Qout        : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END registerM ;

ARCHITECTURE behavior OF registerM IS
BEGIN
  PROCESS(Clk,RST,LDM)
  BEGIN
  IF(RST='0') Then
     IF(Clk 'EVENT and Clk='1') Then
            IF(LDM='1') Then
                Qout <= D ;
                END IF ;
         END IF ;
  ELSIF(RST='1') Then
  Qout <= "00000000" ;
  END IF ;
  END PROCESS ;
END behavior ;
```

REGISTER Q

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

ENTITY registerQ IS
PORT ( RST,LDQ     : IN STD_LOGIC ;
       SL , SR     : IN STD_LOGIC ;
               SIL , SIR   : IN STD_LOGIC ;
               SOL , SOR   : OUT STD_LOGIC ;
       D           : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
               Clk  , DIV1 : IN STD_LOGIC;
               Qout        : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END registerQ ;

ARCHITECTURE behavior OF registerQ IS

SIGNAL qtry : STD_LOGIC_VECTOR(7 downto 0);
```

```
BEGIN

              SOL     <= qtry(7) ;
              SOR     <= qtry(0);
          Qout(7) <= qtry(7);
          Qout(6) <= qtry(6);
              Qout(5) <= qtry(5);
              Qout(4) <= qtry(4);
              Qout(3) <= qtry(3);
              Qout(2) <= qtry(2);
              Qout(1) <= qtry(1);
              Qout(0) <= qtry(0);

  PROCESS(Clk,RST,LDQ)
  BEGIN

  IF(RST='0') Then


         IF(Clk 'EVENT and Clk='1') Then
            IF(LDQ='1') Then
                qtry <= D ;

                ELSIF(SL='1' AND LDQ='0') Then

                qtry(7) <= qtry(6);
                qtry(6) <= qtry(5);
                qtry(5) <= qtry(4);
                qtry(4) <= qtry(3);
                qtry(3) <= qtry(2);
                qtry(2) <= qtry(1);
                qtry(1) <= qtry(0);
                qtry(0) <= SIL ;

                ELSIF(SR='1' AND LDQ='0' AND SL='0') Then
                qtry(7) <= SIR;
                qtry(6) <= qtry(7);
                qtry(5) <= qtry(6);
                qtry(4) <= qtry(5);
                qtry(3) <= qtry(4);
                qtry(2) <= qtry(3);
                qtry(1) <= qtry(2);
                qtry(0) <= qtry(1);

                ELSIF(DIV1='1') Then
                qtry(0) <= '1' ;


                END IF;
         END IF ;
  ELSIF(RST='1') Then
  qtry <= "00000000" ;
  END IF ;
  END PROCESS ;
END behavior ;
```

REGISTER QM1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

ENTITY registerQm1 IS
PORT ( RST,LDQ     : IN STD_LOGIC ;
       SL , SR     : IN STD_LOGIC ;
               SIR         : IN STD_LOGIC ;
               Clk         : IN STD_LOGIC;
               SOL         : OUT STD_LOGIC ;
           Qm1out        : BUFFER STD_LOGIC);

END registerQm1 ;

ARCHITECTURE behavior OF registerQm1 IS



BEGIN

  PROCESS(Clk,RST,LDQ)
  BEGIN

  IF(RST='0') Then
     SOL <= Qm1out ;
         IF(Clk 'EVENT and Clk='1') Then
            IF(LDQ='1') Then
                Qm1out <= '0' ;

                ELSIF(SL='1' AND LDQ='0') Then

                Qm1out <= '0' ;

                ELSIF(SR='1' AND LDQ='0' AND SL='0') Then
                Qm1out <= SIR ;

                END IF;
         END IF ;
  ELSIF(RST='1') Then
  Qm1out <= '0' ;
  END IF ;
  END PROCESS ;
END behavior ;
```

REGISTER FILE (RF)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity RF is
   generic (l : INTEGER := 8; -- l: number of registers in RF
            w : INTEGER := 8; -- w: register width (or # bits in a register)
            m : INTEGER := 3); -- m: # bits in register address
```

```
 Port ( RST, WE : in STD_LOGIC;
      SrcAdrA, SrcAdrB, DstAdr : in STD_LOGIC_VECTOR(m-1 DOWNTO 0);
      clk : in STD_LOGIC;
      DataIn : in STD_LOGIC_VECTOR(w-1 DOWNTO 0);
      DataOutA, DataOutB : out STD_LOGIC_VECTOR(w-1 DOWNTO 0));
end RF;


architecture Behavioral of RF is
-- following are defined for the outputs of the D-FF array
signal tmpq: std_logic_vector(w*l-1 downto 0);
-- following are the load signals for individual registers
signal load: std_logic_vector(l-1 downto 0);



component RFregister is
   Port ( RST, LOAD : in STD_LOGIC;
       clk : in STD_LOGIC;
         D : in STD_LOGIC_VECTOR(w-1 DOWNTO 0);
         O : out STD_LOGIC_VECTOR(w-1 DOWNTO 0));
end component;


begin
-- Generation of correct number of registers:
  genreg1: for i in l-1 downto 0 generate begin
             registers: RFregister port map(
                RST => RST,
                LOAD => load(i),
                clk => clk,
                D => DataIn(w-1 DOWNTO 0),
                O => tmpq((i+1)*w-1 DOWNTO i*w)
     );
  end generate genreg1;


-- Parameterized DstAdr Decoder:
p1: process (WE, DstAdr) begin
for i in l-1 downto 0 loop
if ((i=conv_integer('0'&DstAdr)) AND (WE='1')) then
load(i) <= '1';
else
load(i) <= '0';
end if;
end loop;
end process p1;


-- Parameterized SrcAdrA MUX:
p2: process (SrcAdrA) begin
for i in l-1 downto 0 loop
if i=conv_integer('0'&SrcAdrA) then
DataOutA <= tmpq((i+1)*w-1 DOWNTO i*w);
end if;
end loop;
end process p2;


-- Parameterized SrcAdrB MUX:
p3: process (SrcAdrB) begin
```

```
for i in l-1 downto 0 loop
if i=conv_integer('0'&SrcAdrB) then
DataOutB <= tmpq((i+1)*w-1 DOWNTO i*w);
end if;
end loop;
end process p3;
end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity to_7seg is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
          seg7 : out  STD_LOGIC_VECTOR (7 downto 0)
             );
end to_7seg;

architecture Behavioral of to_7seg is

begin

--'a' corresponds to MSB of seg7 and 'g' corresponds to LSB of seg7.
process (A)
BEGIN
    case A is
        when "0000"=> seg7 <="11000000";  -- '0'
        when "0001"=> seg7 <="11111001";  -- '1'
        when "0010"=> seg7 <="10100100";  -- '2'
        when "0011"=> seg7 <="10110000";  -- '3'
        when "0100"=> seg7 <="10011001";  -- '4'
        when "0101"=> seg7 <="10010010";  -- '5'
        when "0110"=> seg7 <="10000010";  -- '6'
        when "0111"=> seg7 <="11111000";  -- '7'
        when "1000"=> seg7 <="10000000";  -- '8'
        when "1001"=> seg7 <="10010000";  -- '9'
        when "1010"=> seg7 <="10001000";  -- 'A'
        when "1011"=> seg7 <="10000011";  -- 'b'
        when "1100"=> seg7 <="11000110";  -- 'C'
        when "1101"=> seg7 <="10100001";  -- 'd'
        when "1110"=> seg7 <="10000110";  -- 'E'
        when "1111"=> seg7 <="10001110";  -- 'F'
        when others =>  NULL;
    end case;
end process;

end Behavioral;
```

BYTE SELECTOR (UPPER OR LOWER)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ByteSelector is
    Port ( SEL : in  STD_LOGIC;
           DataNew    : in  STD_LOGIC_VECTOR(7 downto 0);
           DataOld    : in  STD_LOGIC_VECTOR(7 downto 0);
           CombinedData : out STD_LOGIC_VECTOR(15 downto 0));
end ByteSelector ;

architecture Behavioral of ByteSelector is
```

```
begin
    process(SEL)
               begin
                      case SEL is

                             when '0' =>

                             CombinedData(7 downto 0) <= DataNew;
                             CombinedData(15 downto 8) <= DataOld;

                             when '1' =>

                             CombinedData(15 downto 8) <= DataNew;
                             CombinedData(7 downto 0) <= DataOld;

                             when others =>

                      end case;
               end process;
end Behavioral;
```

8 – BIT DOWN COUNTER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity DOWN_COUNTER is
    Port ( clk : in std_logic; -- clock input
           resetcounter: in std_logic; -- reset input
                        enable : in std_logic;
           counter: out std_logic_vector(3 downto 0) ;-- output 4-bit counter
                        count : out std_logic
     );
end DOWN_COUNTER;

architecture Behavioral of DOWN_COUNTER is
signal counter_down: std_logic_vector(3 downto 0);
begin

-- down counter
process(clk,resetcounter,enable)
begin
if(rising_edge(clk)) then
    if(resetcounter='0') then
         counter_down <= x"8";
    elsif(resetcounter='1' and enable='1') then
        counter_down <= counter_down - x"1";
    end if;
 end if;
end process;
 counter <= counter_down;


 process(counter_down)
         begin
         case counter_down  is

          when  x"0" =>
              count <= '0' ;
              when others  =>
              count <='1' ;
              end case ;
end process ;



end Behavioral;
```

REGISTER AM1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

ENTITY registerAm1 IS
PORT ( RST,LDAmsb,Ain  : IN STD_LOGIC ;
               Clk         : IN STD_LOGIC;
           Qm1out        : BUFFER STD_LOGIC);

END registerAm1 ;

ARCHITECTURE behavior OF registerAm1 IS

BEGIN

  PROCESS(Clk,RST,LDAmsb)
  BEGIN

  IF(RST='0') Then
         IF(Clk 'EVENT and Clk='1') Then
            IF(LDAmsb='1') Then
                Qm1out <= Ain ;

                     ELSE
                     Qm1out <= Qm1out;
                     END IF;
         END IF ;
  ELSIF(RST='1') Then
  Qm1out <= '0' ;

  END IF ;
  END PROCESS ;
END behavior ;
```

MUX 4 TO 1 (9 – BIT OUTPUT PC SELECT)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity mux4to1ninebits is
       port(
              A, B, C, D            : in std_logic_vector(9 downto 0);
              output                : out std_logic_vector(9 downto 0);
              Selectbits            : in std_logic_vector(1 downto 0)
              );
end mux4to1ninebits;


architecture selector of mux4to1ninebits is
begin
              process(Selectbits)
              begin
                     case Selectbits is

                            when "00" => output <= A;

                            when "01" => output <= B;

                            when "10" => output <= C;

                            when "11" => output <= D;

                            when others =>

                     end case;
              end process;
end selector;
```

Assembler

We developed C based assembler for our project, this assembler can give given code output in hexadecimal format which can be used in Quartus software to program CPU.

```
/*


BAD Basic Assembler for Kintel
Created by: Barış Dinç
Instructions Supported : 16

Compiled on GCC 8.1.0
Written on Visial Studio Code



*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int countoccurrences(char *filename, char c);
int getPos(char *token);
int getInt(char *token);
int assembler(char *line, int programCounter, int labelcount, char **labelname, int *labeladdress);
int* decToBinary(int n) ;
int* decToHexa(int n);

//Register table
char registerTable[8][2]={"$0","$1",
                            "$2","$3",
                            "$4","$5","$6","$7",
                            };

//Instruction lookup table
char
nameofinst[15][5]={"add","sub","mult","div","beq","bneq","jmp","sb","lb","movi","addi","subi","slt","lb
u","sbu"};
int number[15]={0,4096,8192,12288,16384,20480,24576,28672,32768,40960,45056,49152,53248,57344,61440};
int type[15]={1,1,2,2,8,8,5,4,4,6,3,3,7,4,4};

int main(){
    int option;
    int labelcount;

    while(option!=3){
        int labelcount,programCounter=0,labelCounter=0;
        int assembledline=0;

        int* labeladdress;
        char **labelname;
        fflush(stdin);
        int* binary;
        int* hexadd;
        char line [ 256 ];
        char line2 [ 256 ];
        printf("\nPlease select program mode:\n1:Interactive Mode\n2:Batch Mode\n3:Exit Program\n\n");
        scanf("%d",&option);

        if(option==1){
            //Reading user entered instruction
            printf("\nPlease enter line to assembly: ");
            scanf("%s %s",line,line2);
            strcat(line, " ");
            strcat(line,line2);

            //Assembler function does binary conversion
            assembledline = assembler(line,programCounter,labelcount,labelname,labeladdress);

            //If wrong instruction sent or instruction does not exist sends error with line with
problem
            if(assembledline==0){
                printf("\nError in instruction\n");
            }
            //assembler function does assembly as integer, dectobinary changes it to binary
            binary = decToHexa(assembledline);
            int j;
            for(j=3;j>-1;j--)
                printf("%c",binary[j]);
            printf("\n");

        }
        else if (option==2){
            char *fileName = (char*)malloc(sizeof(char)*50);
            printf("\nPlease enter file name: ");
            scanf("%s",fileName);

            //Opens user defined file
            FILE *file = fopen ( fileName, "r" );

            if ( file != NULL ){
                // Counts how many labels exist in so it can create a table for their addresses
                labelcount = countoccurrences(fileName,':');

                labelname=(char**)malloc(sizeof(char*)*labelcount);

                for(int i =0; i<labelcount; i++)
                    *(labelname+i)=(char*)malloc(sizeof(char)*50);
                labeladdress = (int*)malloc(sizeof(int)*labelcount);

                //reads file line by line and counts programcounter to save labels to array for later
branch and jump calculations
                while ( fgets ( line, sizeof line, file ) != NULL ){
                    if (strpbrk(line, ":") != NULL){
                        char *t = strtok(line,":");
                        strcpy(labelname[labelCounter],t);
                        labeladdress[labelCounter]=programCounter;
                        labelCounter++;
                    }
                    programCounter+=1;
                }
            }
            else{
                printf("\nFile does not exist!\n");
                continue;
            }
            programCounter=0;

            rewind(file);
            FILE *fout = fopen("out.txt","w");

            //Starts reading file again for assembly process
            if ( file != NULL ) {
                //does conversion line by line with assembler function
                while ( fgets ( line, sizeof line, file ) != NULL ){
                    printf("%s",line);
                    assembledline = assembler(line,programCounter,labelcount,labelname,labeladdress);

                    if(assembledline==0){
                        printf("\nError in line: %d\n",(programCounter/4)+1);
                    }

                    binary = decToHexa(assembledline);
                    hexadd = decToHexa(programCounter);
                    int j;
                    for(j=3;j>-1;j--){

                        fprintf(fout,"%c",hexadd[j]);

                    }
                    fprintf(fout,":");
                    for(j=3;j>-1;j--){

                        fprintf(fout,"%c",binary[j]);
                    }
                    fprintf(fout,";\n");
                    programCounter+=1;
                }
                printf("\nOutput written in out.obj!\n");
                fclose(file);
                fclose(fout);
            }
        }
        else if(option==3)
            exit(1);
        else
            printf("\nOption does not exist!\n");
    }
    return 0;
}

int assembler(char *line,int programCounter, int labelcount, char **labelname, int *labeladdress){
    char *token;
    int reg, assembledline=0;
    int types;
    char instName[5], rd[50], rs[50], rt[50], rdt[50];
    int i,k;
    int status=0;

    /*This function takes instruction and divides it into parts with comma,space between register names
and instruction name etc.
    Then copies these parts into 3 different strings
    Depending on op code it does corresponding allocations to 32 bit assembly line output function
works with extra spaces,tab move
    */

    if ((token=strpbrk(line, ":")) != NULL){
        for(i=0;i<strlen(token);i++)
            token[i]=token[i+1];
        token = strtok(token, " \t\n");
    }
    else
        token = strtok(line, " \t\n");

    strcpy(instName,token);
    int c=1;
    while( token != NULL ) {
        token = strtok(NULL, ", )(\n");
        if(c==1){
            strcpy(rd,token);
        }
        else if(c==2 && token!=NULL){
            strcpy(rs,token);
        }
        else if(c==3 && token!=NULL){
            strcpy(rt,token);
        }
        else if(c==4 && token!=NULL){
            strcpy(rdt,token);
            break;
        }
        c++;
    }
    for(k=0;k<16;k++){
        //Function detects inst name and compares with table defined before depending on function it
allocates corresponding bits and enters register cases
        if(strcmp(instName,nameofinst[k])==0){
            assembledline = number[k];
            types = type[k];

            int lAddress;
            //types here are for different register,label allocations on 32 bit (ex. add and sll are R
type but requires different bitwise operations)
            switch(types){
                case 1:
                    //gets registers and after shifts puts them on their places getPos returns register
number
                    //add,sub,slt
                    reg = getPos(rd);
                    reg = reg<<3;
                    assembledline=assembledline|reg;

                    reg = getPos(rs);
                    reg = reg<<9;
                    assembledline=assembledline|reg;

                    reg = getPos(rt);
                    reg = reg<<6;
                    assembledline=assembledline|reg;

                    break;
                case 2:
                    // for mult and div operations
                    reg = getPos(rd);
                    reg = reg<<3;
                    assembledline=assembledline|reg;

                    reg = getPos(rs);
                    assembledline=assembledline|reg;

                    reg = getPos(rt);
                    reg = reg<<9;
                    assembledline=assembledline|reg;

                    reg = getPos(rdt);
                    reg = reg<<6;
                    assembledline=assembledline|reg;
                    break;
                case 3:
                    //for Immediate
                    reg = getPos(rd);
                    reg = reg<<6;
                    assembledline=assembledline|reg;

                    reg = getPos(rs);
                    reg = reg<<9;
                    assembledline=assembledline|reg;

                    reg = getInt(rt);
                    reg = reg&63;
                    assembledline=assembledline|reg;

                    break;
                case 4:
                    //for Sb and Lb operations
                    reg = getPos(rd);
                    reg = reg<<6;
                    assembledline=assembledline|reg;

                    reg = getInt(rs);
                    reg = reg&63;
                    assembledline=assembledline|reg;

                    reg = getPos(rt);
                    reg = reg<<9;
                    assembledline=assembledline|reg;

                    break;
                case 5:
                //gets label data from previous array, and makes corresponding calculations and bitwise
operations


                    for(i=0;i<labelcount;i++){
                        if(strcmp(labelname[i],rd)==0){

                            lAddress = labeladdress[i];
                            reg = lAddress;
                            reg = reg & 4095;
                            status=1;
                            assembledline = assembledline|reg;
                            break;
                        }
                    }
                    if(status==0){
                        printf("\nNo Label found!\n");
                        assembledline = 0;
                    }

                    break;
                case 6:
                    //for Movi
                    reg = getPos(rd);
                    reg = reg<<9;
                    assembledline=assembledline|reg;

                    reg = getInt(rs);
                    reg = reg<<1;
                    reg = reg & 510;
                    assembledline = assembledline | reg;

                    break;
                case 7:
                    //gets registers and after shifts puts them on their places getPos returns register
number
                    //add,sub,slt
                    reg = getPos(rd);
                    reg = reg << 9;
                    assembledline = assembledline | reg;

                    reg = getPos(rs);
                    reg = reg << 6;
                    assembledline = assembledline | reg;

                    reg = getPos(rt);
                    reg = reg << 3;
                    assembledline = assembledline | reg;

                    break;
                case 8:
                 /* gets label data from previous label array, and makes corresponding calculations and
bitwise operations if no label found,
                 checks if it is a number and writes that number to 32 bit, if case is not any of them
gives no label error
                */
                    reg = getPos(rd);
                    reg = reg<<9;
                    assembledline=assembledline|reg;

                    reg = getPos(rs);
                    reg = reg<<6;
                    assembledline=assembledline|reg;

                    for(i=0;i<labelcount;i++){
                        if(strcmp(labelname[i],rt)==0){
                            lAddress = labeladdress[i];
                            reg = (lAddress-(programCounter+1));
                            reg = reg & 63;
                            status=1;
                            break;
                        }
                    }

                    char *t;
                    int val = strtol(rt,&t,10);

                    if(val !=0 && status==0){
                        reg = getInt(rt);
                    }
                    else if (status==0){
                        printf("\nError in line %d\nNo Label found!\n",(programCounter/4)+1);
                        return 0;
                    }

                    assembledline = assembledline|reg;
                    break;
            }
        }
    }
    return assembledline;
}
// this functions gets register number
int getPos(char *token){
    int i;
    for(i = 0;i<32;i++)
      if(strncmp(token,registerTable[i],2)==0)
        return (long)i;
    return 0;
}
// this function changes char number to integer for instruction
int getInt(char *token){
    char *a;
    int signednumber=0;
    signednumber = strtol(token,&a,10) & 255;
    return signednumber;
}
//counts ':' to find how many labels are exist in the code
int countoccurrences(char *filename, char c){
    int count=0;
    FILE *fp = fopen(filename,"r");
    if (fp){
        int ch;
        while ((ch = fgetc(fp)) != EOF)
            if (ch == c) count++;

        fclose(fp);
    }
    else
        printf("Failed to open");

    return count;
}

int* decToBinary(int n)
{
    // array to store binary number
    int *binaryNum;
    binaryNum=(int*)malloc(sizeof(int)*32);

    // counter for binary array
    int i = 0;

    for(i=0;i<32;i++)
        binaryNum[i]=0;

    i=0;
    if(n<0)
        n=4294967296+n;
    while (n > 0) {
        // storing remainder in binary array
        binaryNum[i] = n % 2;
        n = n / 2;
        i++;
    }

    return binaryNum;
}

// function to convert decimal to hexadecimal
int* decToHexa(int n)
{
    // char array to store hexadecimal number
    int *hexaDeciNum;
    hexaDeciNum=(int*)malloc(sizeof(int)*32);

    // counter for hexadecimal number array
    int i = 0;
    while(n!=0)
    {
        // temporary variable to store remainder
        int temp  = 0;

        // storing remainder in temp variable.
        temp = n % 16;

        // check if temp < 10
        if(temp < 10)
        {
            hexaDeciNum[i] = temp + 48;
            i++;
        }
        else
        {
            hexaDeciNum[i] = temp + 55;
            i++;
        }

        n = n/16;
    }


    return hexaDeciNum;
}
```

Conclusion

In this project we are given with ISA design objectives stating from basic Booths multiplier algorithm. Then using course knowledge and software tools we build up working CPU design and implemented on a FPGA chip. This CPU can run given benchmarks which shows project objectives are completed. All the details about project were discussed in this report.

In the end, this project taught a lot about VHDL, ISA design and how computer architects work. With limited resources we designed 8-bit CPU works in 40 MHz range and this CPU can make a lot of daily operations. Maybe programming will be hard for it because of limited number of instructions but having an assembler can help developers of this CPU a lot. To improve this design, I/O modules can be introduced or methods for clock speed increase can be applied. Also, different working method(Pipeline or von Neumann) can be introduced to the system.