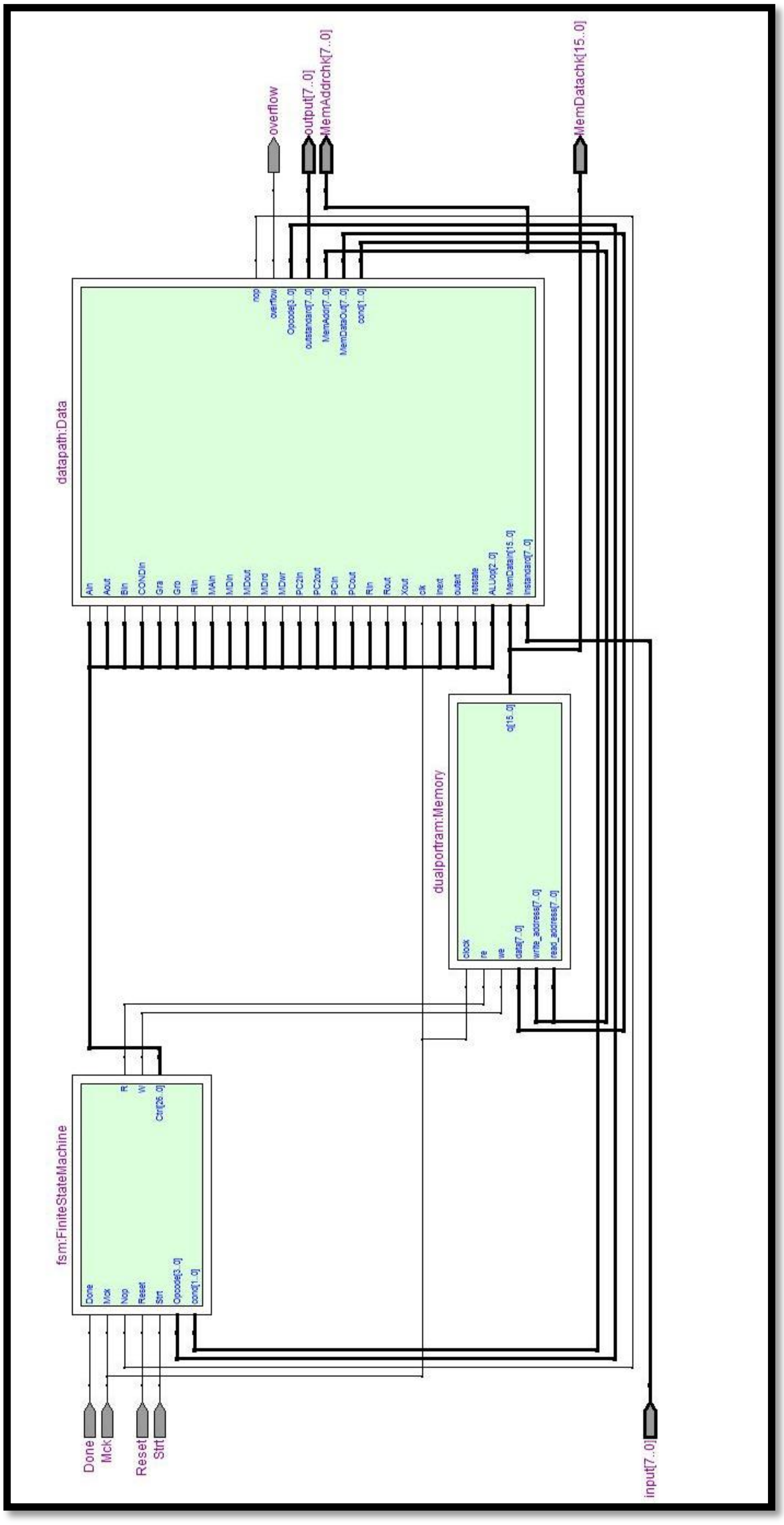
8MP16.



overflow
output[7..0]
MemAddrchk[7..0]
MemDatachk[15..0]
datapath:Data
dualportram:Memory
fsm:FiniteStateMachine
Done
Mck
Reset
Strt
input[7..0]

Abstract RTN

architectural_state :=
(
PC<7..0>:
PC2<7..0>:
IR<15..0>:
R[3..0]<7..0>:
A<7..0> := R[3]<7..0>:
MA<7..0>:
MD<15..0>:
B<7..0>:
COND<1..0>:
OVERF<0>:
OUT<7..0>:
):
memory := (M[255..0]<7..0>:):
ext := {standard input/output}:

instruction_format :=
(
IR<15..12> := b0000:
IR<11..8> := op<3..0>:

(format_1 :=)
IR<7..6> := b00:
IR<5..4> := ra<1..0>:
IR<3..2> := b00:
IR<1..0> := rb<1..0>:

(format_2 :=)
IR<7..0> := X<7..0>:
):

instruction_interpretation := (instruction_fetch; instruction_execute):
instruction_fetch := (IR ← M[PC]; PC ← PC + 4);
instruction_execute :=
(
(type_0 :=)
inc → A ← A + 1:
dec → A ← A — 1:
in → A ← ext:
out → ext ← A:

(type_1 :=)
not → R[ra] = ¬ R[ra]:
read → A ← M[X]:
write → M[X] ← A:

br → PC ← M[PC + X]:
brz → (A = 0 → PC ← M[PC + X]):
brg → (A > 0 → PC ← M[PC + X]):

(type_2 :=)
add → R[ra] ← R[ra] + R[rb]:
sub → R[ra] ← R[ra] — R[rb]:
and → R[ra] ← R[ra] ∧ R[rb]:
or → R[ra] ← R[ra] ˅ R[rb]:
):

Details

This implementation of the 8MP16 uses a single 8-bit bus in the processor. It interfaces with memory using two 16-bit buses for data and one 8-bit for addressing. All arithmetical-logical operations, except for the incrementing of the program counter, stores the result directly to an accumulator register (named Acc). The average number of cycles per instruction is (87 total cycles)/(16 instructions) = 5.44 or about 5 CPI. The period of each cycle is tsu + tco + th = 5.798 ns + 11.384 ns + 0.541 ns = 16.646 ns or about 16.7 ns, and the frequency of the clock is about 1/(16.7 ns) = 59.88 MHz.

The logic for the register file and for the condition register are instantiated separately from their corresponding components in order to accomodate the debugging process. For a similar reason, many of the tri-state buffers that connect each register to the processor bus are instantiated in the top level of the datapath in order to easily check during simulation that only one signal drives the bus at any given time. The signals used for this type of checking were removed after debugging.

The program counter (PC) and condition register (CONDreg) are set to zero by the reset (rststate) control signal. Signals Ain, PCin, PC2in, Bin, CONDin, IRin, and MAin serve as the "load" signals for their corresponding registers, while such signals as Aout, PCout, and PC2out enable tri-state buffers that connect registers to the processor bus. Xout enables the buffer that outputs the lower 8 bits of IR during instructions that use direct or relative addressing. Signals inext and outext enable buffers that pass values from standard input/output to the processor bus. Rin loads the register file, while Rout is ANDed with the output of the register-select logic (rflogic) to enable the buffers that connect the register file to the bus. Gra and Grb are used by rflogic to gate either the ra or rb fields from IR into the decoder that selects a register from the register file.

The memory data register (MD) is 16 bits wide in order to accomodate instructions, but when receiving from the accumulator or the data memory, it only accepts 8-bit data. To accomplish this, the processor bus is connected to its upper 8 bits of input, with the lower 8 padded by zeros (unused). Only its upper 8 bits are output onto the processor bus (through a tri-state buffer named MDcpubuf). The instruction register (IR) receives its upper 8 bits from the bus and its lower 8 bits directly from MD (through the signal named IRextend). The accumulator can only receive the upper 8 bits of MD through the ALU's PASS operation. The memory can only receive the upper 8 bits of MD during the write instruction, as memory has a word-width of a single byte and can only write once per cycle. However, two bytes may be read and stored to MD from memory within a single cycle. (A Wait signal is necessary in the control unit to prevent the next state from arriving during this cycle to allow time for MD to be written.) The control signal MDin loads a value from the processor bus into MD, while the signal MDrd loads a value from the memory bus. MDout enables a tri-state buffer (MDcpubuf) which outputs to the processor bus, while MDwr enables a tri-state buffer (MDmembuf) which outputs to the memory bus.

To prevent the accumulator from being overwritten during each fetch stage, a separate register (named PC2) is used to store the incremented value of the PC, as well as an adder that is separate from the ALU. In the case of a branch instruction, however, the branch target must be computed by the ALU and so, it is stored to the accumulator.

The single-bus architecture requires that a register (B) be used to store one operand during two-operand instructions such as ADD, SUB, AND, and OR. In these cases, the second operand is accessed from the register file, which has three ports: address, input data, and output data. Both data ports are connected to the processor bus. During branch instructions (relative addressing), the second operand is received from the lower 8 bits of IR. This field of IR is labelled X.

The 2-bit condition register stores a 1 in bit zero if the accumulator contains a zero and a 1 in bit one if the accumulator contains a number greater than zero (signed). The standard output instruction uses a register (outreg) that stores the value of the accumulator in order to maintain the value on the output device for as many cycles as desired.

Notes

Although four bits in each instruction are labelled as the opcode, an extra fifth bit is output from the datapath (named nop) to accomodate a nop instruction. This bit is included only for the sake of completeness. A state and control word in the FSM already exists for nop due to the delay slot that occurs after a branch instruction fails.

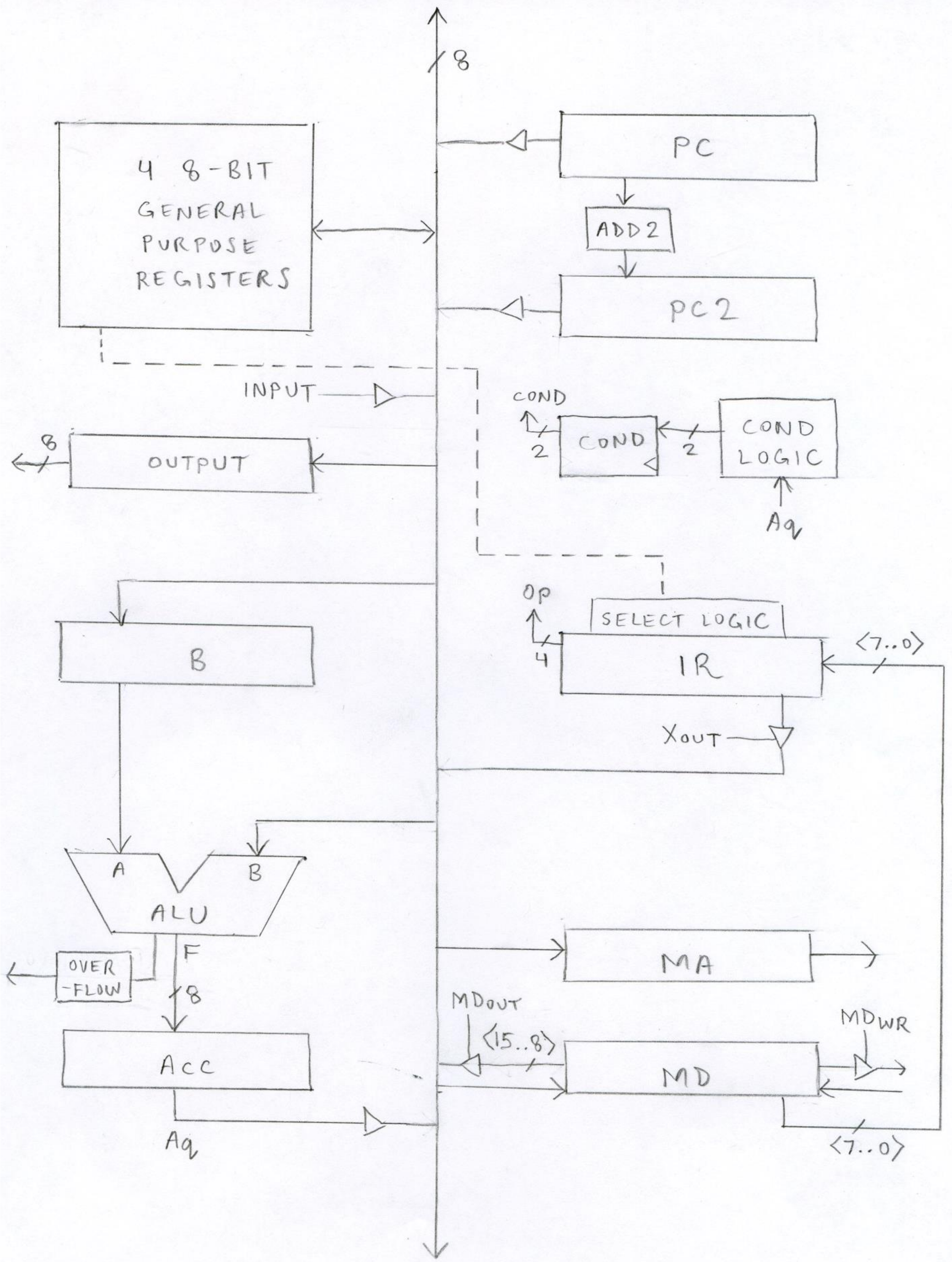
The output signals MemAddrchk and MemDatachk are only intended to aid in tracking the progress of instruction execution. They are not crucial to the design.

VHDL

```
-- Top-Level
LIBRARY IEEE; USE IEEE.STD_LOGIC_1164.ALL;
ENTITY Final_8MP16 IS
PORT (  Mck, Strt, Reset, Done:  IN STD_LOGIC;
        input:                   IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        output:                  OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        overflow:                OUT STD_LOGIC;
        MemDatachk:              OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        MemAddrchk:              OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END Final_8MP16;

ARCHITECTURE Final_8MP16arch OF Final_8MP16 IS

SIGNAL Opcode:                   STD_LOGIC_VECTOR(3 downto 0);
SIGNAL cond:                     STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL R, W, Nop:                STD_LOGIC;
SIGNAL Control:                  STD_LOGIC_VECTOR(26 DOWNTO 0);
SIGNAL DataRd:                   STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL DataWr, MemAddr:          STD_LOGIC_VECTOR(7 DOWNTO 0);

COMPONENT fsm IS PORT (
Opcode:                          IN STD_LOGIC_VECTOR(3 downto 0);
cond:                            IN STD_LOGIC_VECTOR(1 DOWNTO 0);
Done, Mck, Strt, Reset, Nop:     IN STD_LOGIC;
Ctrl:                            OUT STD_LOGIC_VECTOR(26 DOWNTO 0);
```

```
R, W:                           OUT STD_LOGIC);
END COMPONENT;

COMPONENT datapath IS PORT (
ALUop:                          IN STD_LOGIC_VECTOR(2 DOWNTO 0);
clk, rststate, PC2in, Ain, Aout, Bin, CONDin, Gra, Grb, IRin,
MAin, MDin, MDout, MDrd, MDwr, PC2out, PCin, PCout,
Rin, Rout, Xout, inext, outext: IN STD_LOGIC;
Opcode:                         OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
MemDataIn:                      IN STD_LOGIC_VECTOR(15 DOWNTO 0);
instandard:                     IN STD_LOGIC_VECTOR(7 DOWNTO 0);
outstandard:                    OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
MemAddr:                        OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
MemDataOut:                     OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
cond:                           OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
overflow, nop:                  OUT STD_LOGIC);
END COMPONENT;

COMPONENT dualportram IS
GENERIC(        address_width   : INTEGER := 8;
                data_width      : INTEGER := 8);
PORT(           clock           : IN  STD_LOGIC;
                data            : IN  STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);
                write_address   : IN  STD_LOGIC_VECTOR(address_width-1 DOWNTO 0);
                read_address    : IN  STD_LOGIC_VECTOR(address_width-1 DOWNTO 0);
                re, we          : IN  STD_LOGIC;
                q               : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END COMPONENT;

BEGIN
FiniteStateMachine: fsm PORT MAP(     Opcode, cond, Done, Mck, Strt, Reset,
                                      Nop, Control, R, W);
Data: datapath PORT MAP(              Control(23 DOWNTO 21), Mck, Control(26),
                                      Control(20), Control(19), Control(18),
                                      Control(17), Control(16), Control(15),
                                      Control(14), Control(13), Control(12),
                                      Control(11), Control(10), Control(9),
                                      Control(8), Control(7), Control(6),
                                      Control(5), Control(4), Control(3),
                                      Control(2), Control(1), Control(0),
                                      Opcode, DataRd, input, output, MemAddr,
                                      DataWr, cond, overflow, Nop);
Memory: dualportram PORT MAP(         Mck, DataWr, MemAddr, MemAddr,
                                      R, W, DataRd);
MemDatachk    <= DataRd;
MemAddrchk    <= MemAddr;
END Final_8MP16arch;
```

Datapath.


8
PC
ADD2
PC2
4 8-BIT GENERAL PURPOSE REGISTERS
INPUT
COND
2
COND
2
COND LOGIC
Aq
8
OUTPUT
OP
SELECT LOGIC
4
IR
⟨7..0⟩
XOUT
B
A
B
ALU
F
OVER-FLOW
8
MA
MDOUT
⟨15..8⟩
MDWR
ACC
MD
Aq
⟨7..0⟩

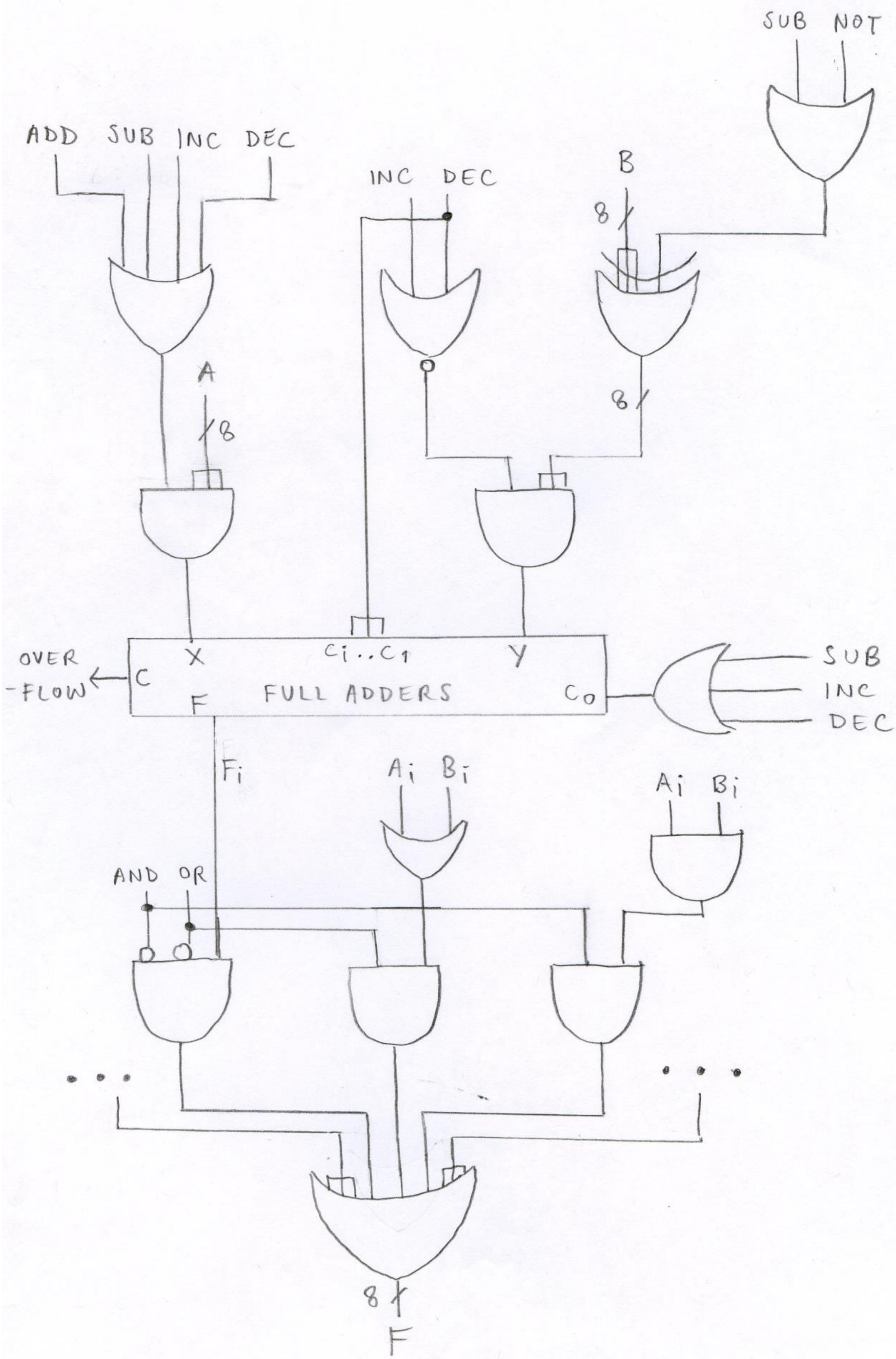
Datapath

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY datapath IS PORT (
ALUop:                          IN STD_LOGIC_VECTOR(2 DOWNTO 0);
clk, rststate, PC2in, Ain, Aout, Bin, CONDin, Gra, Grb, IRin,
MAin, MDin, MDout, MDrd, MDwr, PC2out, PCin, PCout,
Rin, Rout, Xout, inext, outext: IN STD_LOGIC;
Opcode:                         OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
MemDataIn:                      IN STD_LOGIC_VECTOR(15 DOWNTO 0);
instandard:                     IN STD_LOGIC_VECTOR(7 DOWNTO 0);
outstandard:                    OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
MemAddr:                        OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
MemDataOut:                     OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
cond:                           OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
overflow, nop:                  OUT STD_LOGIC);
END datapath;

ARCHITECTURE datapathstruct OF datapath IS
COMPONENT TriState8 IS
GENERIC(        width: INTEGER := 8);
PORT(           E: IN STD_LOGIC;
                D: IN STD_LOGIC_VECTOR(width-1 DOWNTO 0);
                Y: OUT STD_LOGIC_VECTOR(width-1 DOWNTO 0));
END COMPONENT;
COMPONENT TriState16 IS
GENERIC(        width: INTEGER := 16);
PORT(           E: IN STD_LOGIC;
                D: IN STD_LOGIC_VECTOR(width-1 DOWNTO 0);
                Y: OUT STD_LOGIC_VECTOR(width-1 DOWNTO 0));
END COMPONENT;
COMPONENT ff IS
PORT(   clock, Load, D: IN STD_LOGIC;
        Q: OUT STD_LOGIC);
END COMPONENT;
COMPONENT reg8 IS
GENERIC(        width: INTEGER := 8);
PORT(           clock, Load: IN STD_LOGIC;
                D: IN STD_LOGIC_VECTOR(width-1 DOWNTO 0);
                Q: OUT STD_LOGIC_VECTOR(width-1 DOWNTO 0));
END COMPONENT;
COMPONENT reg16 IS
GENERIC(        width: INTEGER := 16);
PORT(           clock, Load: IN STD_LOGIC;
                D: IN STD_LOGIC_VECTOR(width-1 DOWNTO 0);
                Q: OUT STD_LOGIC_VECTOR(width-1 DOWNTO 0));
END COMPONENT;
```

```
COMPONENT pc IS
GENERIC(        width: INTEGER := 8);
PORT(           clock, Clear, Load: IN STD_LOGIC;
                D: IN STD_LOGIC_VECTOR(width-1 DOWNTO 0);
                Q: OUT STD_LOGIC_VECTOR(width-1 DOWNTO 0));
END COMPONENT;
COMPONENT MD IS
GENERIC(        width: INTEGER := 16);
PORT(           clock, MDin, MDrd: IN STD_LOGIC;
                Dcpu, Dmem: IN STD_LOGIC_VECTOR(width-1 DOWNTO 0);
                MDq: OUT STD_LOGIC_VECTOR(width-1 DOWNTO 0));
END COMPONENT;
COMPONENT regfile IS
GENERIC(        width: INTEGER := 8);
PORT(           clock, RE, WE: IN STD_LOGIC;
                addr: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                indata: IN STD_LOGIC_VECTOR(width-1 DOWNTO 0);
                outdata: OUT STD_LOGIC_VECTOR(width-1 DOWNTO 0));
END COMPONENT;
COMPONENT rflogic IS
GENERIC(        width: INTEGER := 2);
PORT(           gra, grb: IN STD_LOGIC;
                ra, rb: IN STD_LOGIC_VECTOR(width-1 DOWNTO 0);
                regsel: OUT STD_LOGIC_VECTOR(width-1 DOWNTO 0));
END COMPONENT;
COMPONENT add2 IS
GENERIC(        width: INTEGER := 8);
PORT (          A: IN STD_LOGIC_VECTOR(width-1 DOWNTO 0);
                F: OUT STD_LOGIC_VECTOR(width-1 DOWNTO 0));
END COMPONENT;
COMPONENT ALU_8MP16 IS
GENERIC(        width: INTEGER := 8);
PORT (          S: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
                A, B: IN STD_LOGIC_VECTOR(width-1 DOWNTO 0);
                F: OUT STD_LOGIC_VECTOR(width-1 DOWNTO 0);
over: OUT STD_LOGIC);
END COMPONENT;
COMPONENT ffcond IS
GENERIC(        width: INTEGER := 2);
PORT(           Clock, Clear, Load: IN STD_LOGIC;
                D: IN STD_LOGIC_VECTOR(width-1 DOWNTO 0);
                Q: OUT STD_LOGIC_VECTOR(width-1 DOWNTO 0));
END COMPONENT;
COMPONENT condlogic IS
PORT(           acc: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                brzcond, brgcond: OUT STD_LOGIC);
END COMPONENT;
```

```
SIGNAL MDq, IRq, MDextend, IRextend:        STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL cpubus, PCq, PC2q, PC2d, Aq, Bq, F:  STD_LOGIC_VECTOR(7 downto 0);
SIGNAL rfaddr, CONDd:                       STD_LOGIC_VECTOR(1 downto 0);
SIGNAL overf:                               STD_LOGIC;

BEGIN
MDextend    <= cpubus & "00000000";
IRextend    <= cpubus & MDq(7 DOWNTO 0);

ProgCount:      pc PORT MAP(clk, rststate, PCin, cpubus, PCq);
ProgCountNext:  reg8 PORT MAP(clk, PC2in, PC2d, PC2q);
PCadd:          add2 PORT MAP(PCq, PC2d);
MAreg:          reg8 PORT MAP(clk, MAin, cpubus, MemAddr);
Acc:            reg8 PORT MAP(clk, Ain, F, Aq);
B:              reg8 PORT MAP(clk, Bin, cpubus, Bq);
MDreg:          MD PORT MAP(clk, MDin, MDrd, MDextend, MemDataIn, MDq);
IR:             reg16 PORT MAP(clk, IRin, IRextend, IRq);
RFlog:          rflogic PORT MAP(Gra, Grb, IRq(5 DOWNTO 4), IRq(1 DOWNTO 0), rfaddr);
RF:             regfile PORT MAP(clk, Rout, Rin, rfaddr, cpubus, cpubus);
ArithLogUnit:   ALU_8MP16 PORT MAP(ALUop, Bq, cpubus, F, overf);
CONDreg:        ffcond PORT MAP(clk, rststate, CONDin, CONDd, cond);
CondLog:        condlogic PORT MAP(cpubus, CONDd(0), CONDd(1));
outreg:         reg8 PORT MAP(clk, outext, cpubus, outstandard);
overff:         ff PORT MAP(clk, Ain, overf, overflow);

PCbuf:      TriState8 PORT MAP(PCout, PCq, cpubus);
PC2buf:     TriState8 PORT MAP(PC2out, PC2q, cpubus);
Accbuf:     TriState8 PORT MAP(Aout, Aq, cpubus);
MDmembuf:   TriState8 PORT MAP(MDwr, MDq(15 DOWNTO 8), MemDataOut);
MDcpubuf:   TriState8 PORT MAP(MDout, MDq(15 DOWNTO 8), cpubus);
inbuf:      TriState8 PORT MAP(inext, instandard, cpubus);
Xbuf:       TriState8 PORT MAP(Xout, IRq(7 DOWNTO 0), cpubus);

Opcode <= IRq(11 DOWNTO 8);
nop    <= IRq(12);
END datapathstruct;
```

ALU.


SUB
NOT
ADD
SUB
INC
DEC
INC
DEC
B
8
A
8
8
OVER
-FLOW
C
X
Ci..C1
Y
F
FULL ADDERS
C0
SUB
INC
DEC
Fi
Ai
Bi
Ai
Bi
AND
OR
8
F

.

ALU

The ALU receives two signed 8-bit values, A and B, and outputs to a signed signal named F. In the datapath, the A input is driven by the B register, and the B input is driven by the processor bus. The B input is driven by either a value from the register file or from the lower 8 bits of IR (passed onto the bus by the tri-state buffer named Xbuf). The ALU operation is selected by three bits (S):

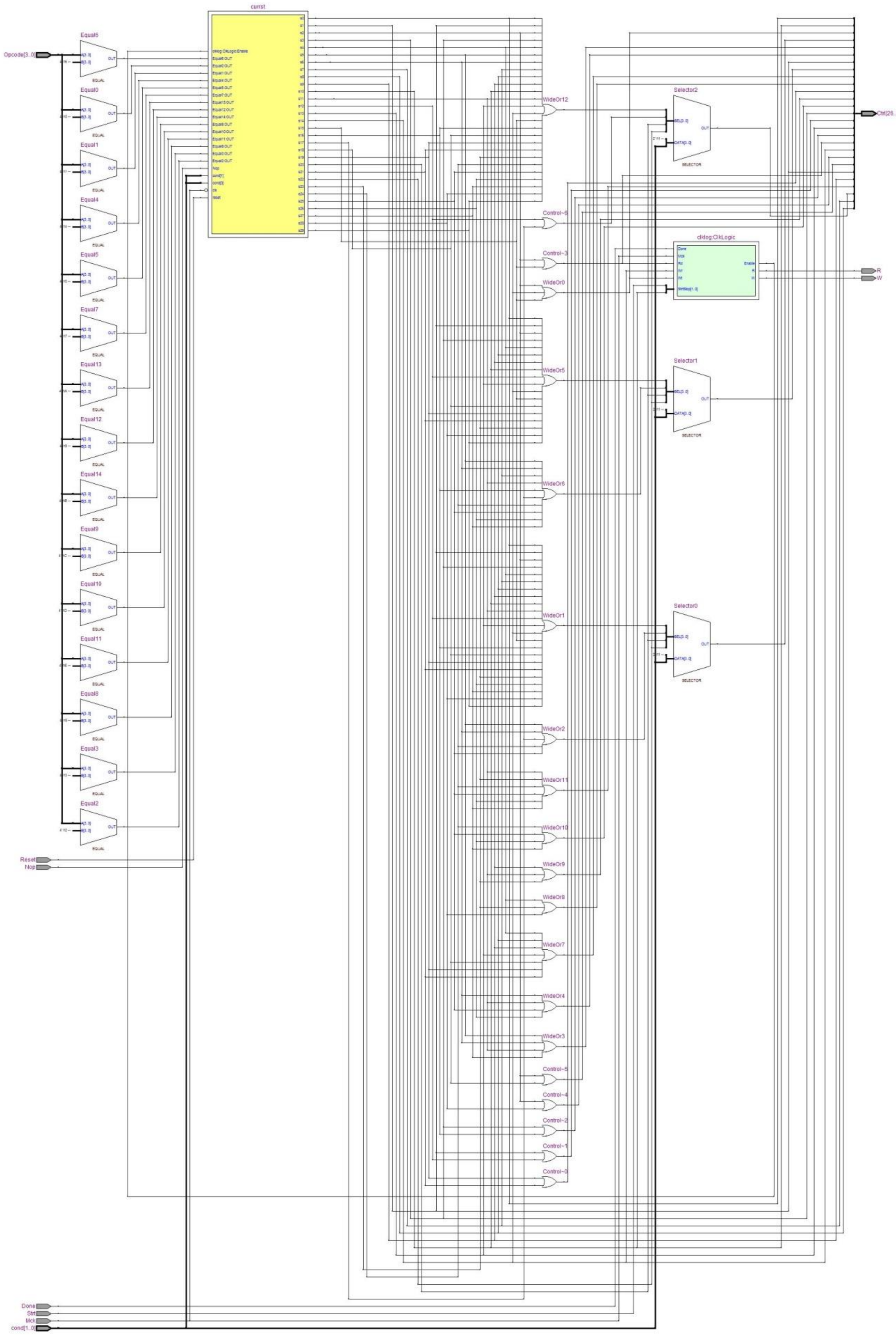
Encoding	Mnemonic	Operation
000	PASS	pass B
001	AND	A AND B
010	OR	A OR B
011	NOT	NOT B
100	ADD	A + B
101	SUB	A - B
110	INC	increment A
111	DEC	decrement A

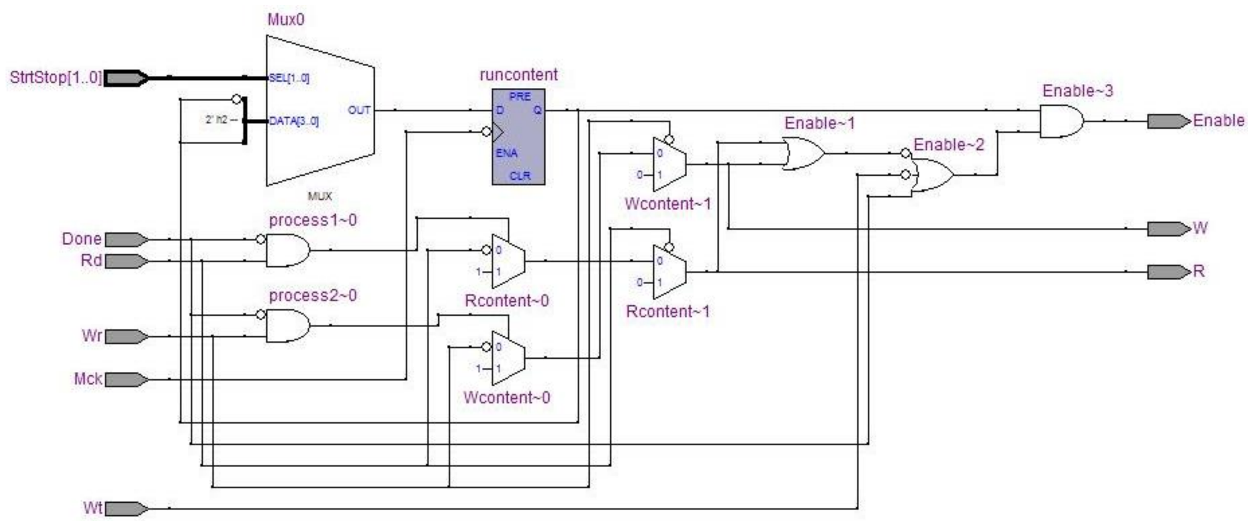
A ninth output bit represents the overflow condition and is saved to a flag flip-flop (named overff in the top-level datapath) whose value is visible in the top level of the processor.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY ALU_8MP16 IS
GENERIC(        width: INTEGER := 8);
PORT (          S: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
                A, B: IN STD_LOGIC_VECTOR(width-1 DOWNTO 0);
                F: OUT STD_LOGIC_VECTOR(width-1 DOWNTO 0);
                over: OUT STD_LOGIC);
END ALU_8MP16;
ARCHITECTURE alustruct OF ALU_8MP16 IS

COMPONENT adders IS
GENERIC(        width: INTEGER := 8);
PORT (          x, y, Ci:  IN STD_LOGIC_VECTOR(width-1 DOWNTO 0);
                Cout:    OUT STD_LOGIC;
                SUM:     OUT STD_LOGIC_VECTOR(width-1 DOWNTO 0));
END COMPONENT;

SIGNAL SUBINC:                                    STD_LOGIC;
SIGNAL faout:                                     STD_LOGIC_VECTOR(width DOWNTO 0);
SIGNAL ASID, SNID, NEGv:                          STD_LOGIC_VECTOR(width-1 DOWNTO 0);
SIGNAL AND_AB, OR_AB, NOT_B, ADD, SUB, INC, DEC:  STD_LOGIC;

BEGIN
AND_AB          <= NOT S(2) AND NOT S(1) AND S(0);
OR_AB           <= NOT S(2) AND S(1) AND NOT S(0);
NOT_B           <= NOT S(2) AND S(1) AND S(0);
```

```
ADD          <= S(2) AND NOT S(1) AND NOT S(0);
SUB          <= S(2) AND NOT S(1) AND S(0);
INC          <= S(2) AND S(1) AND NOT S(0);
DEC          <= S(2) AND S(1) AND S(0);

FAin1: FOR i IN width-1 DOWNTO 0 GENERATE
ASID(i)  <= (ADD OR SUB OR INC OR DEC) AND A(i);
END GENERATE FAin1;

FAin2: FOR j IN width-1 DOWNTO 0 GENERATE
SNID(j)  <= ( (SUB OR NOT_B) XOR B(j) ) AND ( NOT(INC OR DEC) );
END GENERATE FAin2;

NEGin: FOR m IN width-1 DOWNTO 1 GENERATE
NEGv(m)      <= DEC;
END GENERATE NEGin;

NEGv(0)      <= SUB OR INC OR DEC;

RippleCarry: adders PORT MAP( ASID, SNID, NEGv,
                     faout(width), faout(width-1 DOWNTO 0));

over    <= faout(width);

output: FOR z IN width-1 DOWNTO 0 GENERATE
F(z)    <=   (NOT AND_AB AND NOT OR_AB AND faout(z)) OR
             (AND_AB AND A(z) AND B(z)) OR
             (OR_AB AND (A(z) OR B(z)));
END GENERATE output;
END alustruct;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY adders IS
GENERIC(     width:   INTEGER := 8);
PORT (       x, y, Ci:  IN STD_LOGIC_VECTOR(width-1 DOWNTO 0);
             Cout:    OUT STD_LOGIC;
             SUM:     OUT STD_LOGIC_VECTOR(width-1 DOWNTO 0));
END adders;

ARCHITECTURE Structural OF adders IS
BEGIN
SUM <= x + y + Ci;
Cout <= (x(width-1) AND y(width-1)) OR
        (Ci(width-1) AND (x(width-1) XOR y(width-1)));
END Structural;
```

FSM.


currst
Opcode[3..0]
Equal6
Equal0
Equal1
Equal4
Equal5
Equal7
Equal13
Equal12
Equal14
Equal9
Equal10
Equal11
Equal8
Equal3
Equal2
Reset
Nop
Done
Str
Mck
cond[1..0]
WideOr12
Selector2
Ctrl[26..0]
Control~6
clklog:ClkLogic
Control~3
WideOr0
R
W
WideOr5
Selector1
WideOr6
WideOr1
Selector0
WideOr2
WideOr11
WideOr10
WideOr9
WideOr8
WideOr7
WideOr4
WideOr3
Control~5
Control~4
Control~2
Control~1
Control~0



Clocking Logic.

<u>Control</u>

The clocking logic (clklogic) supervises the FSM and its interaction with memory. It generates the "wait" signal (Wt) that pauses the FSM until the Done signal indicates that memory access has completed. It maintains the "read" and "write" signals (R and W) that allow memory access to continue during the cycles when Done has not yet been asserted. It also generates the Run signal that drives the Enable signal, which in turn permits the FSM to proceed to the next state.

The FSM defines its state sequence and control words according to the steps in the concrete RTN description. It returns to the reset state (s0) when the external Reset signal is asserted, during which the rststate signal is sent to the datapath, clearing the PC and condition register. If Enable is asserted by the clocking logic, the FSM enters the next state on the falling edge of the clock.

<u>Control Signals</u>

26	rststate
25	stop
24	Wt

23	ALUop2
22	ALUop1
21	ALUop0
20	PC2in
19	Ain
18	Aout
17	Bin
16	CONDin

15	Gra
14	Grb
13	IRin
12	MAin
11	MDin
10	MDout
9	MDrd
8	MDwr

7	PC2out
6	PCin
5	PCout
4	Rin
3	Rout
2	Xout
1	inext
0	outext

<u>States</u>

State	RTN	Signals	Control Word
s0		rststate	100 0000 0000 0000 0000 0000 0000
s1	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in	000 0001 0000 0001 0000 0010 0000
s2	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out	001 0000 0000 0000 0010 1100 0000
s3	IR ← MD;	IRin, MDout	000 0000 0000 0010 0100 0000 0000
s4	B ← A;	Bin, Aout	000 0000 0110 0000 0000 0000 0000
s5	A ← B+1;	Ain, INC1	000 1100 1000 0000 0000 0000 0000
s6	A ← B−1;	Ain, DEC1	000 1110 1000 0000 0000 0000 0000
s7	A ← ext;	Ain, inext, PASS	000 0000 1000 0000 0000 0000 0010
s8	ext ← A;	Aout, outext	000 0000 0100 0000 0000 0000 0001
s9	R[ra] ← A;	Rin, Gra, Aout	000 0000 0100 1000 0000 0001 0000
s10		Stop	010 0000 0000 0000 0000 0000 0000
s11	A ← ¬R[ra];	Ain, Rout, Gra, NOT	000 0110 1000 1000 0000 0000 1000
s12	MA ← X;	MAin, Xout	000 0000 0000 0001 0000 0000 0100
s13	MD ← A;	MDin, Aout	000 0000 0100 0000 1000 0000 0000
s14	M[MA] ← MD;	Write, Wait	001 0000 0000 0000 0001 0000 0000
s15	MD ← M[MA];	Read, Wait	001 0000 0000 0000 0010 0000 0000
s16	A ← MD;	Ain, MDout, PASS	000 0000 1000 0000 0100 0000 0000
s17	A ← B+X;	Ain, Xout, ADD	000 1000 1000 0000 0000 0000 0100
s18	PC ← A;	PCin, Aout	000 0000 0100 0000 0000 0100 0000
s19	A=0 → COND<0> ← 1: A!=0 → COND<0> ← 0;	Aout, CONDin	000 0000 0101 0000 0000 0000 0000
s20	COND<0>=1 → A ← B+X;	COND<0> → (Ain, Xout, ADD)	000 1000 1000 0000 0000 0000 0100
s21	A>0 → COND<1> ← 1: A<=0 → COND<1> ← 0;	Aout, CONDin	000 0000 0101 0000 0000 0000 0000
s22	COND<1>=1 → A ← B+X;	COND<1> → (Ain, Xout, ADD)	000 1000 1000 0000 0000 0000 0100
s23	B ← R[ra];	Bin, Rout, Gra	000 0000 0010 1000 0000 0000 1000
s24	A ← B+R[rb];	Ain, Rout, Grb, ADD	000 1000 1000 0100 0000 0000 1000
s25	A ← B−R[rb];	Ain, Rout, Grb, SUB	000 1010 1000 0100 0000 0000 1000
s26	A ← B ∧ R[rb];	Ain, Rout, Grb, AND	000 0010 1000 0100 0000 0000 1000
s27	A ← B ∨ R[rb];	Ain, Rout, Grb, OR	000 0100 1000 0100 0000 0000 1000
s28	B ← PC2;	Bin, PC2out	000 0000 0010 0000 0000 1000 0000
s29		nop	000 0000 0000 0000 0000 0000 0000

Concrete RTN

Type 0:

inc (op := x00)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3	B ← A;	Bin, Aout
T4	A ← B+1;	Ain, INC1

dec (op := x01)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3	B ← A;	Bin, Aout
T4	A ← B—1;	Ain, DEC1

in (op := x02)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3	A ← ext;	Ain, inext, PASS

out (op := x03)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3	ext ← A;	Aout, outext

mva (op := x0E)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3	R[ra] ← A;	Rin, Gra, Aout

stop (op := x0F)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3		Stop

Type 1:

not (op := x08)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3	A ← ¬R[ra];	Ain, Rout, Gra, NOT
T4	R[ra] ← A;	Rin, Gra, Aout

write (op := x0C)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3	MA ← X;	MAin, Xout
T4	MD ← A;	MDin, Aout
T5	M[MA] ← MD;	Write, Wait

read (op := x0D)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3	MA ← X;	MAin, Xout
T4	MD ← M[MA];	Read, Wait
T5	A ← MD;	Ain, MDout, PASS

br (op := x09)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3	B ← PC2;	Bin, PC2out
T4	A ← B+X;	Ain, Xout, ADD
T5	PC ← A;	PCin, Aout

brz (op := x0A)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3	B ← PC2;	Bin, PC2out
T4	A=0 → COND<0> ← 1: A!=0 → COND<0> ← 0;	Aout, CONDin
T5	COND<0>=1 → A ← B+X;	COND<0> → (Ain, Xout, ADD)
T6	PC ← A;	PCin, Aout

brg (op := x0B)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3	B ← PC2;	Bin, PC2out
T4	A>0 → COND<1> ← 1: A<=0 → COND<1> ← 0;	Aout, CONDin
T5	COND<1>=1 → A ← B+X;	COND<1> → (Ain, Xout, ADD)
T6	PC ← A;	PCin, Aout

Type 2:

add (op := x04)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3	B ← R[ra];	Bin, Rout, Gra
T4	A ← B+R[rb];	Ain, Rout, Grb, ADD
T5	R[ra] ← A;	Rin, Gra, Aout

sub (op := x05)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3	B ← R[ra];	Bin, Rout, Gra
T4	A ← B—R[rb];	Ain, Rout, Grb, SUB
T5	R[ra] ← A;	Rin, Gra, Aout

and (op := x06)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3	B ← R[ra];	Bin, Rout, Gra
T4	A ← B ∧ R[rb];	Ain, Rout, Grb, AND
T5	R[ra] ← A;	Rin, Gra, Aout

or (op := x07)

T0	MA ← PC: PC2 ← PC+2;	MAin, PCout, PC2in
T1	MD ← M[MA]: PC ← PC2;	Read, Wait, PCin, PC2out
T2	IR ← MD;	IRin, MDout
T3	B ← R[ra];	Bin, Rout, Gra
T4	A ← B ˅ R[rb];	Ain, Rout, Grb, OR
T5	R[ra] ← A;	Rin, Gra, Aout

VHDL

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY fsm IS PORT (    Opcode:                      IN STD_LOGIC_VECTOR(3 downto 0);
                        cond:                        IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                        Done, Mck, Strt, Reset, Nop: IN STD_LOGIC;
                        Ctrl:                        OUT STD_LOGIC_VECTOR(26 DOWNTO 0);
                        R, W:                        OUT STD_LOGIC);
END fsm;

ARCHITECTURE fsmbehav OF fsm IS
COMPONENT clklog IS PORT (  Done, Mck, Rd, Wt, Wr: IN STD_LOGIC;
                            StrtStop:              IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                            Enable, R, W:          OUT STD_LOGIC);
END COMPONENT;

SIGNAL StrtStop:        STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL En:              STD_LOGIC;
SIGNAL Control:         STD_LOGIC_VECTOR(26 DOWNTO 0);

TYPE st_type IS (       s0, s1, s2, s3, s4, s5, s6, s7, s8, s9,
                        s10, s11, s12, s13, s14, s15, s16, s17,
                        s18, s19, s20, s21, s22, s23, s24, s25,
                        s26, s27, s28, s29);
SIGNAL currst, nextst:  st_type;

BEGIN
StrtStop        <= Strt & Control(25);
ClkLogic:       clklog PORT MAP(Done, Mck, Control(9), Control(24), Control(8), StrtStop, En, R, W);
Ctrl            <= Control;

PROCESS(Mck,Reset,En,currst)
BEGIN
IF Reset='1' THEN
        currst <= s0;
ELSIF En='0' THEN
        currst <= currst;
ELSIF Mck'EVENT AND Mck='0' THEN
        currst <= nextst;
END IF;
END PROCESS;

PROCESS(currst,Opcode,cond,Nop)
BEGIN
CASE currst IS

WHEN s0 =>
```

```
Control <= "100000000000000000000000000";
nextst <= s1;

WHEN s1 =>
Control <= "000000100000001000000100000";
nextst <= s2;

WHEN s2 =>
Control <= "001000000000000001011000000";
nextst <= s3;

WHEN s3 =>
Control <= "000000000000010010000000000";
IF Nop='1' THEN
nextst <= s29;
ELSIF Opcode="0000" OR Opcode="0001" THEN
nextst <= s4;
ELSIF Opcode="0010" THEN
nextst <= s7;
ELSIF Opcode="0011" THEN
nextst <= s8;
ELSIF Opcode="0100" OR Opcode="0101" OR Opcode="0110" OR Opcode="0111" THEN
nextst <= s23;
ELSIF Opcode="1000" THEN
nextst <= s11;
ELSIF Opcode="1001" OR Opcode="1010" OR Opcode="1011" THEN
nextst <= s28;
ELSIF Opcode="1100" OR Opcode="1101" THEN
nextst <= s12;
ELSIF Opcode="1110" THEN
nextst <= s9;
ELSE  --op 1111
nextst <= s10;
END IF;

WHEN s4 =>
Control <= "000000001100000000000000000";
IF Opcode="0000" THEN
nextst <= s5;
ELSE  --op 0001
nextst <= s6;
END IF;

WHEN s5 =>
Control <= "000110010000000000000000000";
nextst <= s1;

WHEN s6 =>
```

```
Control <= "000111010000000000000000000";
nextst <= s1;

WHEN s7 =>
Control <= "000000010000000000000000010";
nextst <= s1;

WHEN s8 =>
Control <= "000000001000000000000000001";
nextst <= s1;

WHEN s9 =>
Control <= "000000001001000000000010000";
nextst <= s1;  --op 1000, 0100, 0101, 0110, 0111, 1110

WHEN s10 =>  --Stop
Control <= "010000000000000000000000000";
nextst <= s10;

WHEN s11 =>
Control <= "000011010001000000000001000";
nextst <= s9;

WHEN s12 =>
Control <= "000000000000001000000000100";
IF Opcode="1100" THEN
nextst <= s13;
ELSE  --op 1101
nextst <= s15;
END IF;

WHEN s13 =>
Control <= "000000001000000100000000000";
nextst <= s14;

WHEN s14 =>
Control <= "001000000000000000100000000";
nextst <= s1;

WHEN s15 =>
Control <= "001000000000000001000000000";
nextst <= s16;

WHEN s16 =>
Control <= "000000010000000010000000000";
nextst <= s1;

WHEN s17 =>
```

```
Control <= "0001000100000000000000000100";
nextst <= s18;

WHEN s18 =>
Control <= "0000000010000000000001000000";
nextst <= s1;

WHEN s19 =>
Control <= "0000000010100000000000000000";
nextst <= s20;

WHEN s20 =>
IF cond(0)='1' THEN
Control <= "0001000100000000000000000100";
nextst <= s18;
ELSE
Control <= "0000000000000000000000000000";
nextst <= s29;
END IF;

WHEN s21 =>
Control <= "0000000010100000000000000000";
nextst <= s22;

WHEN s22 =>
IF cond(1)='1' THEN
Control <= "0001000100000000000000000100";
nextst <= s18;
ELSE
Control <= "0000000000000000000000000000";
nextst <= s29;
END IF;

WHEN s23 =>
Control <= "0000000001010000000000001000";
IF Opcode="0100" THEN
nextst <= s24;
ELSIF Opcode="0101" THEN
nextst <= s25;
ELSIF Opcode="0110" THEN
nextst <= s26;
ELSE  --op 0111
nextst <= s27;
END IF;

WHEN s24 =>
Control <= "0001000100000100000000001000";
nextst <= s9;
```

```
WHEN s25 =>
Control <= "0001010100001000000000001000";
nextst <= s9;

WHEN s26 =>
Control <= "0000010100001000000000001000";
nextst <= s9;

WHEN s27 =>
Control <= "0000100100001000000000001000";
nextst <= s9;

WHEN s28 =>
Control <= "0000000001000000000010000000";
IF Opcode="1001" THEN
nextst <= s17;
ELSIF Opcode="1010" THEN
nextst <= s19;
ELSIF Opcode="1011" THEN
nextst <= s21;
ELSE  --to account for all cases
nextst <= s0;
END IF;

WHEN s29 =>
Control <= "0000000000000000000000000000";
nextst <= s1;
END CASE;
END PROCESS;
END fsmbehav;

-- Clocking Logic
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY clklog IS PORT (  Done, Mck, Rd, Wt, Wr: IN STD_LOGIC;
                         StrtStop: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                         Enable, R, W: OUT STD_LOGIC);
END clklog;

ARCHITECTURE clklogbehav OF clklog IS
SIGNAL Rcontent, Wcontent, runcontent, donecontent, Rq, Wq, SDone, Run: STD_LOGIC;
BEGIN

PROCESS(Done,donecontent)
BEGIN
donecontent <= Done;
END PROCESS;
```

```
SDone <= donecontent;

PROCESS(Rd,SDone,Rcontent)
BEGIN
IF Rd='0' THEN
        Rcontent <= '0';
ELSIF Rd='1' AND SDone='0' THEN
        Rcontent <= '1';
ELSE
        Rcontent <= NOT Rd;
END IF;
END PROCESS;
Rq <= Rcontent;

PROCESS(Wr,SDone,Wcontent)
BEGIN
IF Wr='0' THEN
        Wcontent <= '0';
ELSIF Wr='1' AND SDone='0' THEN
        Wcontent <= '1';
ELSE
        Wcontent <= NOT Wr;
END IF;
END PROCESS;
Wq <= Wcontent;

PROCESS(Mck,StrtStop,runcontent)
BEGIN
IF Mck'EVENT AND Mck='0' THEN
        CASE StrtStop IS
        WHEN "00" =>
                NULL;
        WHEN "01" =>
                runcontent <= '0';
        WHEN "10" =>
                runcontent <= '1';
        WHEN OTHERS =>
                runcontent <= NOT runcontent;
END CASE;
END IF;
END PROCESS;
Run     <= runcontent;
R       <= Rq;
W       <= Wq;
Enable  <= Run AND (SDone OR NOT Wt OR NOT(Rq OR Wq));
END clklogbehav;
```

Memory

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
ENTITY dualportram IS
GENERIC(        address_width: INTEGER := 8;
                data_width:    INTEGER := 8);
PORT(           clock:         IN  STD_LOGIC;
                data:          IN  STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);
                write_address: IN  STD_LOGIC_VECTOR(address_width-1 DOWNTO 0);
                read_address:  IN  STD_LOGIC_VECTOR(address_width-1 DOWNTO 0);
                re, we:        IN  STD_LOGIC;
                q:             OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END dualportram;

ARCHITECTURE rtl OF dualportram IS
TYPE ram IS ARRAY(0 TO 2**address_width-1) OF STD_LOGIC_VECTOR(data_width-1 DOWNTO 0);
SIGNAL          ram_block:                    ram;
ATTRIBUTE       ram_init_file:                STRING;
ATTRIBUTE       ram_init_file OF ram_block:   SIGNAL IS "Test_mv_mem_arithlog.mif";
BEGIN
PROCESS (clock)
BEGIN

IF RISING_EDGE(clock) THEN
        IF (we='1') THEN
                ram_block(TO_INTEGER(UNSIGNED(write_address))) <= data;
        END IF;
        IF (re='1') THEN
                q(15 DOWNTO 8) <= ram_block(TO_INTEGER(UNSIGNED(read_address)));
                q(7 DOWNTO 0) <= ram_block(TO_INTEGER(UNSIGNED(read_address))+1);
        END IF;
END IF;
END PROCESS;
END rtl;
```

Simulation

Timing Analyzer Summary

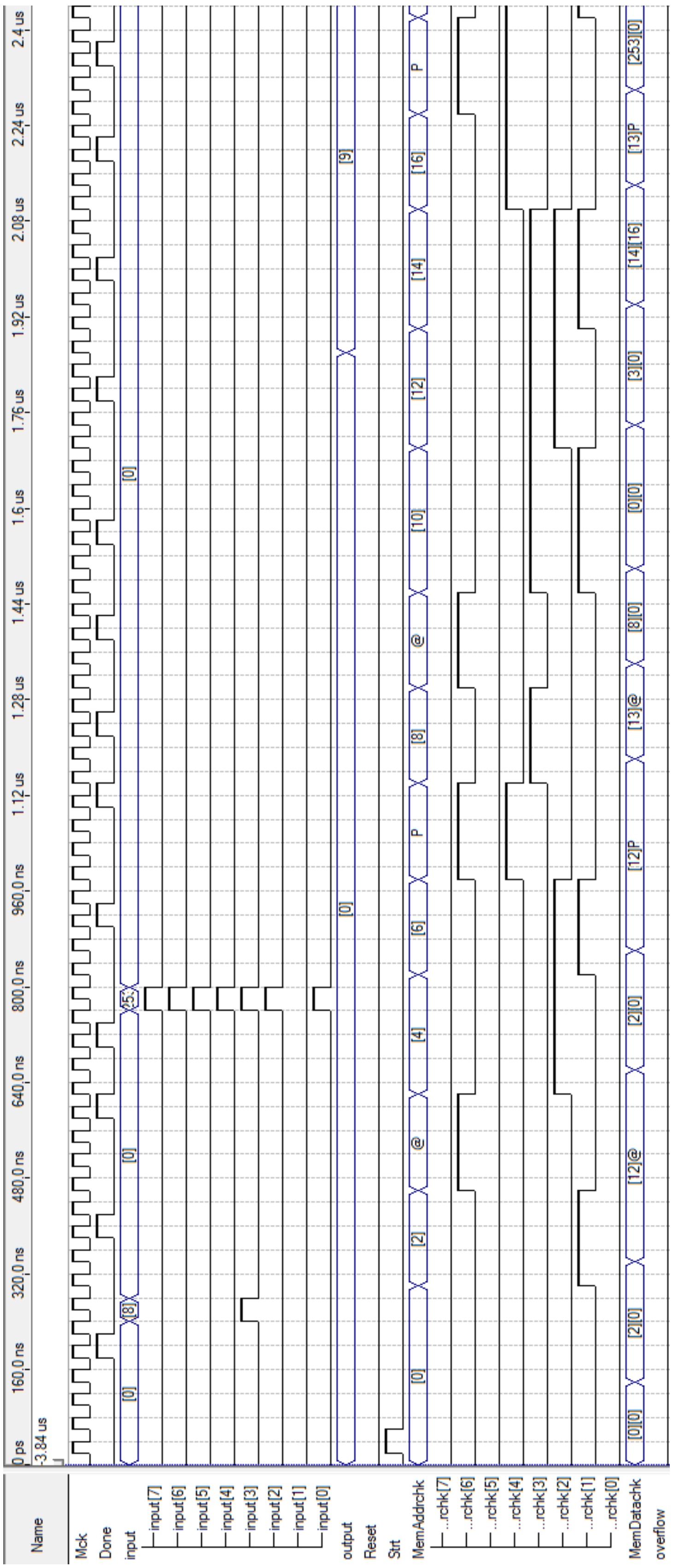
	Type	Slack	Required Time	Actual Time
1	Worst-case tsu	N/A	None	5.798 ns
2	Worst-case tco	N/A	None	11.384 ns
3	Worst-case th	N/A	None	-0.541 ns
4	Clock Setup: 'Mck'	N/A	None	60.07 MHz (period = 16.646 ns)
5	Total number of failed paths			

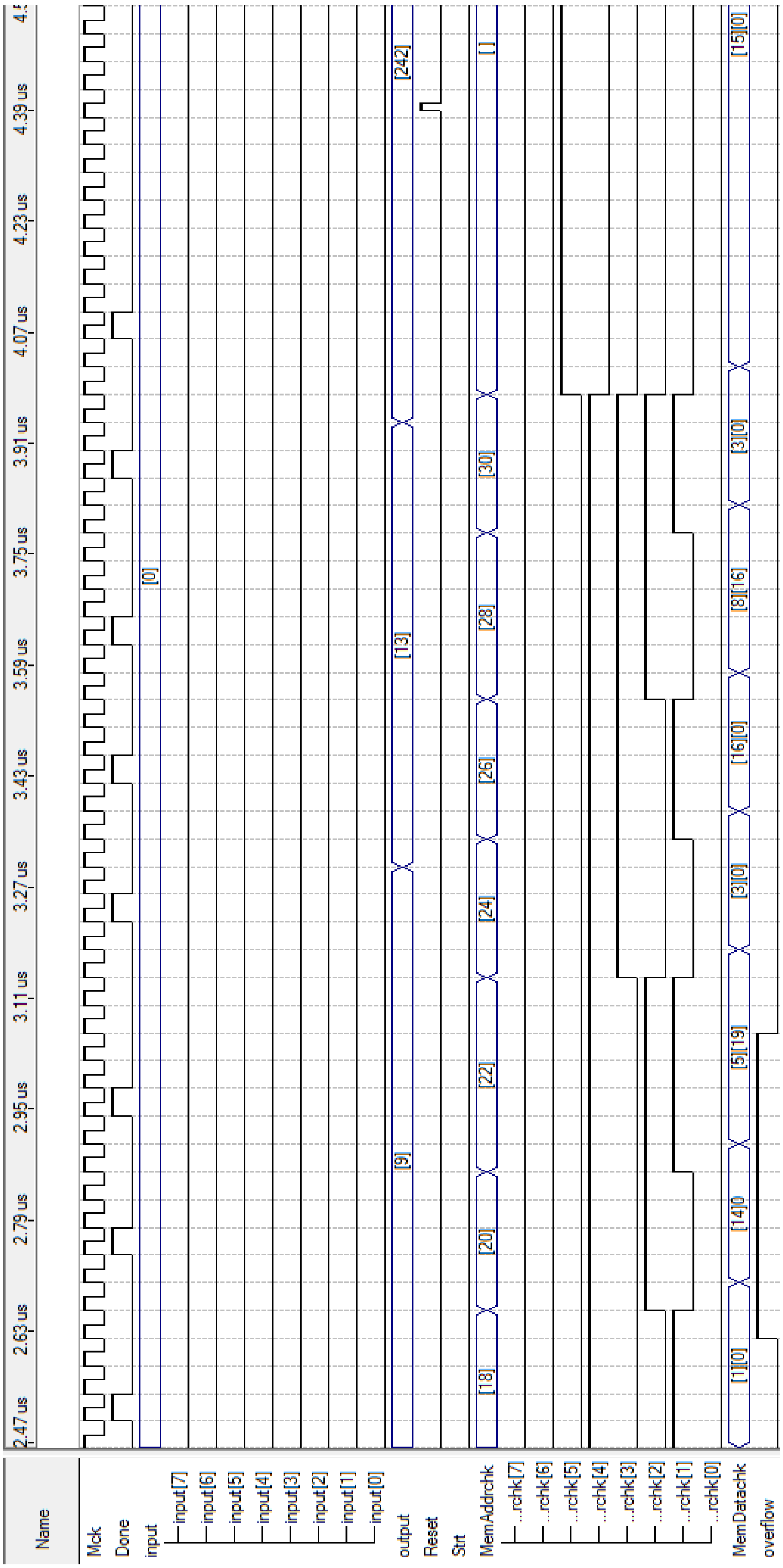
<table>
<tr><th>From</th></tr>
<tr><td>input[2]</td></tr>
<tr><td>dualportram:Memory|altsyncram:ram_block__dual_rtl_0|altsyncram_a7j1:auto_generated|altsyncram_cqm1:altsyncram1|ram_block2a0~porta_address_reg7</td></tr>
<tr><td>input[3]</td></tr>
<tr><td>fsm:FiniteStateMachine|currst.s9</td></tr>
<tr><td></td></tr>
</table>

<table>
<tr><th>To</th><th>From Clock</th><th>To Clock</th><th>Failed Paths</th></tr>
<tr><td>datapath:Data|reg8:Acc|content[7]</td><td>--</td><td>Mck</td><td>0</td></tr>
<tr><td>MemDatachk[7]</td><td>Mck</td><td>--</td><td>0</td></tr>
<tr><td>datapath:Data|reg8:outreg|content[3]</td><td>--</td><td>Mck</td><td>0</td></tr>
<tr><td>datapath:Data|reg8:Acc|content[7]</td><td>Mck</td><td>Mck</td><td>0</td></tr>
<tr><td></td><td></td><td></td><td>0</td></tr>
</table>

Move, Memory Access, and Arithmetic-Logic Test

`in 8          ;A <= 8`	0200
`wr x40        ;M[x40] <= 8`	0C40
`in -3         ;A <= -3`	0200
`wr x50        ;M[x50] <= -3`	0C50
`rd x40        ;A <= 8`	0D40
`inc           ;A <= 9`	0000
`out           ;output 9`	0300
`mva 1         ;R[r1] <= 9`	0E10
`rd x50        ;A <= -3`	0D50
`dec           ;A <= -4`	0100
`mva 3         ;R[r3] <= -4`	0E30
`sub r1, r3    ;R[r1] <= 13`	0513
`out           ;output 13`	0300
`nop`	1000
`not r1        ;R[r1] <= 242`	0810
`out           ;output 242`	0300
`stp`	0F00

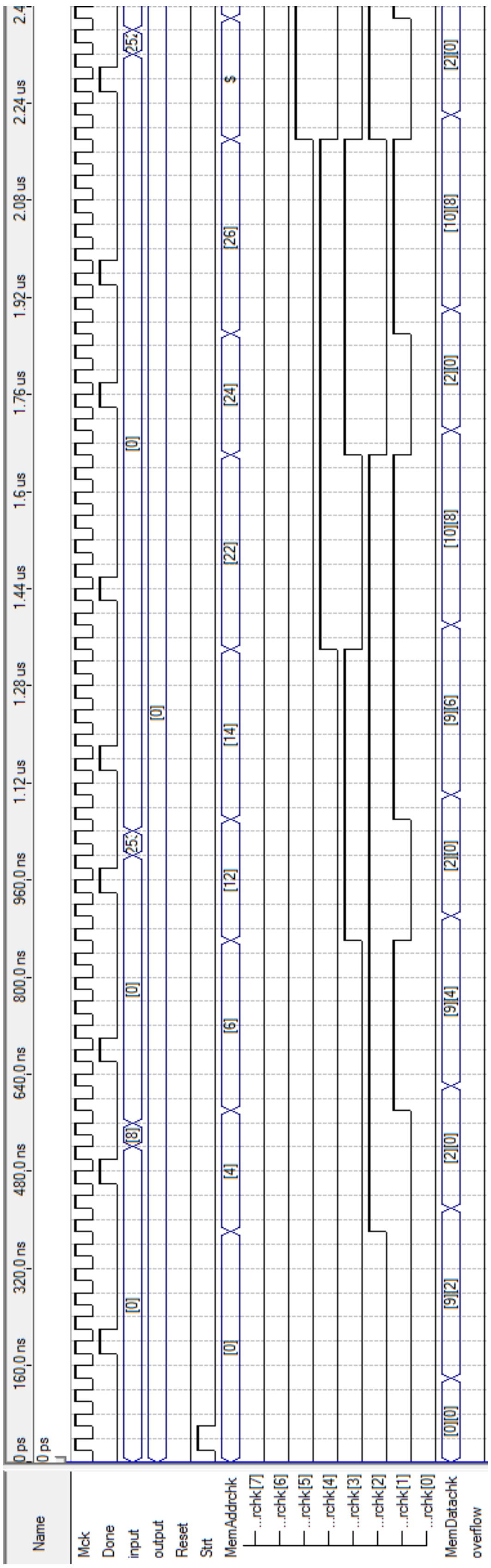

Name
Mck
Done
input
input[7]
input[6]
input[5]
input[4]
input[3]
input[2]
input[1]
input[0]
output
Reset
Strt
MemAddrchk
...rchk[7]
...rchk[6]
...rchk[5]
...rchk[4]
...rchk[3]
...rchk[2]
...rchk[1]
...rchk[0]
MemDatachk
overflow
0 ps
-3.84 us
160.0 ns
320.0 ns
480.0 ns
640.0 ns
800.0 ns
960.0 ns
1.12 us
1.28 us
1.44 us
1.6 us
1.76 us
1.92 us
2.08 us
2.24 us
2.4 us

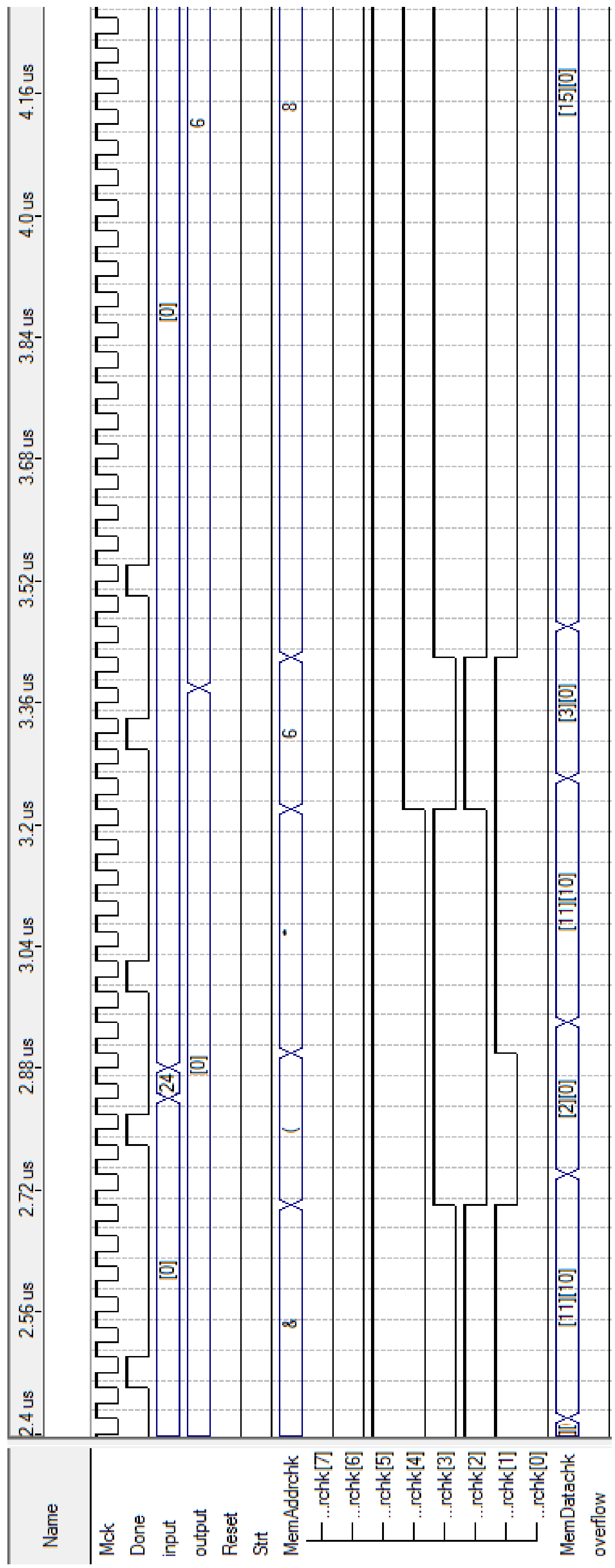


Name
Mck
Done
input
input[7]
input[6]
input[5]
input[4]
input[3]
input[2]
input[1]
input[0]
output
Reset
Strt
MemAddrchk
...rchk[7]
...rchk[6]
...rchk[5]
...rchk[4]
...rchk[3]
...rchk[2]
...rchk[1]
...rchk[0]
MemDatachk
overflow
2.47 us
2.63 us
2.79 us
2.95 us
3.11 us
3.27 us
3.43 us
3.59 us
3.75 us
3.91 us
4.07 us
4.23 us
4.39 us
[0]
[9]
[13]
[242]
[18]
[20]
[22]
[24]
[26]
[28]
[30]
[1][0]
[14][0]
[5][19]
[3][0]
[16][0]
[8][16]
[15][0]

Branch Test

x00	d000	br 4	;PC <= PC2 + 2	0902
x04	d004	in 8	;A <= 8	0200
x06	d006	br 12	;PC <= PC2 + 4	0904
x0C	d012	in -3	;A <= -3	0200
x0E	d014	br 22	;PC <= PC2 + 6	0906
x16	d022	brz 32	;branch failure	0A08
x18	d024	in 0	;A <= 0	0200
x1A	d026	brz 36	;PC <= PC2 + 8	0A08
x24	d036	in -4	;A <= -4	0200
x26	d038	brg 50	;branch failure	0B0A
x28	d040	in 24	;A <= 24	0200
x2A	d042	brg 54	;PC <= PC2 + 10	0B0A
x36	d054	out	;output 54	0300
x38	d056	stp		0F00


Name
Mck
Done
input
output
Reset
Strt
MemAddrchk
...rchk[7]
...rchk[6]
...rchk[5]
...rchk[4]
...rchk[3]
...rchk[2]
...rchk[1]
...rchk[0]
MemDatachk
overflow
0 ps
160.0 ns
320.0 ns
480.0 ns
640.0 ns
800.0 ns
960.0 ns
1.12 us
1.28 us
1.44 us
1.6 us
1.76 us
1.92 us
2.08 us
2.24 us
2.4


Name
Mck
Done
input
output
Reset
Strt
MemAddrchk
...rchk[7]
...rchk[6]
...rchk[5]
...rchk[4]
...rchk[3]
...rchk[2]
...rchk[1]
...rchk[0]
MemDatachk
overflow
2.4 us
2.56 us
2.72 us
2.88 us
3.04 us
3.2 us
3.36 us
3.52 us
3.68 us
3.84 us
4.0 us
4.16 us