Course: Computer Architecture

Exercise: 2

Date: 2018-11-29

Author: Uddeshya Sinha, 2397282S

Author: Peiqi Yu, 2358806Y

Both parts of the exercise are completed, compiled and successfully executed.

Both parts work correctly.

PART I:

The **LoadxiRun.hs** is a machine language program that calculates the sum of the elements of an array with 7 elements. The traditional **load** opcode loads an array element from the memory into the registers and in order to find the sum of all elements of an array, the program needs to iterate over the array and each iteration requires to load an element to a register and then increment the index so as to get the next element. This method is very tedious and hence, a new opcode **loadxi** is introduced to carry out both loading of data (array elements) and incrementing of index in a single instruction rather than two.

The **loadxi** has 4 states from st_loadxi0 being the initial to st_loadxi3 being the final state. The **loadxi** instruction is presented as opcode number 7 in **Control.hs.**

```
st_loadxi0 = dff (pRX!!7)                -- loadxi added as opcode 7
st_loadxi1 = dff st_loadxi0
st_loadxi2 = dff st_loadxi1
st_loadxi3 = dff st_loadxi2
```

The following test data is provided to inspect the correctness of program:

```
                    -- 0014
                    -- 0014   ;Data Area
     "0007",        -- 0014   n         data    7 - total size of array
     "0000",        -- 0015   sum       data    0
     "fffd",        -- 0016   x         data   -3
     "0008",        -- 0017             data    8
     "0004",        -- 0018             data    4
     "0015",        -- 0019             data   21
     "fffa",        -- 001a             data   -6
     "0004",        -- 001b             data    4
     "0005"         -- 001c             data    5
  ]
```

The total sum (in decimal) = 33

The total sum (in hexadecimal) = 21

The first instruction: lea R1, 1[R0] loads the constant 1 in R1:

```
************************************************************************
Executed instruction:  lea  R1,0001[R0]   effective address = 0001
R1 := 0001 was loaded in cycle 33
Processor state:    pc = 0002  ir = f100  ad = 0001
************************************************************************

........................................................................
Clock cycle 34
```

Since i<n (7), the jumpf instruction did not perform jump:

```
jumpf instruction did not perform jump
************************************************************************
Executed instruction:  jumpf  R5,0016[R0]   effective address = 0016
did not jump in cycle 53
Processor state:    pc = 000a  ir = f504  ad = 0016
************************************************************************

........................................................................
```

An instance of the program when the total sum was 001e or 30 in decimal:

```
Register file update: R4 := 001e

************************************************************************

Executed instruction:  add  R4,R4,R6
R4 := 001e was loaded in cycle 106
Processor state:    pc = 000e  ir = 0446  ad = 0019
************************************************************************

........................................................................
```

Finally, the program calculates the sum of array elements which is 21 in hexadecimal:

```
Memory store:  mem[0015] := 0021
************************************************************************
Executed instruction:  store  R4,0015[R0]   effective address = 0015
mem[0015] := 0021 was stored in cycle 191
Processor state:    pc = 0013  ir = f402  ad = 0015
************************************************************************

........................................................................
```

PART II

The **MulRun.hs** is a machine language program for calculating multiplication. Add the multiply function unit to the M1 circuit. When multiplying, the instruction will take a variable number of clock cycles, depending on the sizes of the numbers being multiplied.

The mul has 5 states from st_mul0 being the initial to s4 being the final state. Since the instruction will take a variable number of clock cycles, a condition ready to determine the end of the loop is added to states s2 and s4. Ready is transmit from the Datapath to the control through the M1 circuit to determine if the loop is terminated. When the functional unit has produced a result, the ready changes from 0 to 1, the state changes to s4 and the value of prod will transmit to the register. The mul instruction is presented as opcode number 2 in Control.hs.

```
st_mul0  = dff (pRRR!!2)
s1 = dff st_mul0
s2 = dff (and3 (inv st_mul0) (or2 s1 s3) (inv ready1))
s3 = dff (and2 (inv st_mul0) s2)
s4 = dff (and3 (inv st_mul0) (or2 s1 s3) ready1)
```

The following test data is provided to inspect the correctness of program:

```
"fc00", "0025",    -- 0000  start lea   R12,25[R0]     ; R12 =25
"f300", "0027",    -- 0002        lea  R3,27[R0]       ; R3 = 27
"24c3",            -- 0004        mul R4,R12,R3        ; R4 = :R12 * R3
"fd00", "0005",    -- 0006        lea   R13,5[R0]      ; R13 =5
"f500", "0017",    -- 0008        lea  R5,17[R0]       ; R5 = 17
"26d5",            -- 0010        mul R6,R13,R5        ; R6 = :R13 * R5
"fe00", "0085",    -- 0012        lea   R14,85[R0]     ; R14 =85
"f700", "0097",    -- 0014        lea  R7,97[R0]       ; R7 = 97
"28e7",            -- 0016        mul R8,R14,R7        ; R8 = :R14 * R7
"d000"             -- 0018        trap  R0,R0,R0       ; terminate
```

It has 3 times multiplication:

mul R4, R12, R3, R4 := R12*R3, R12 = 25, R3 = 27

mul R6, R13, R5, R6 := R13*R5, R13 = 5, R5= 17

mul R8, R14, R7, R8 := R14*R7, R14 = 85, R7 = 97

simulation output (mul R8, R14, R7):

lea R14, 85[R0] loads the constant 85 in R14:

```
*************************************************************************
Executed instruction:  lea  R14,0085[R0]   effective address = 0085
R14 := 0085 was loaded in cycle 56
Processor state:    pc = 000c  ir = fe00  ad = 0085
*************************************************************************
```

lea R7, 97[R0] loads the constant 97 in R7:

```
*************************************************************************
Executed instruction:  lea  R7,0097[R0]   effective address = 0097
R7 := 0097 was loaded in cycle 60
Processor state:    pc = 000e  ir = f700  ad = 0097
*************************************************************************
```

Finally, the program calculates the Product of R14 and R7 in hexadecimal:

```
*************************************************************************
Executed instruction:  mul  R8,R14,R7
R8 := 4e73 was loaded in cycle 73

Processor state:    pc = 000f  ir = 28e7  ad = 0097
*************************************************************************
```