ECEC 355
Drexel University, Winter 2021
2/5/2021

David Schmidt
Damodhar Thota
Mark Carson

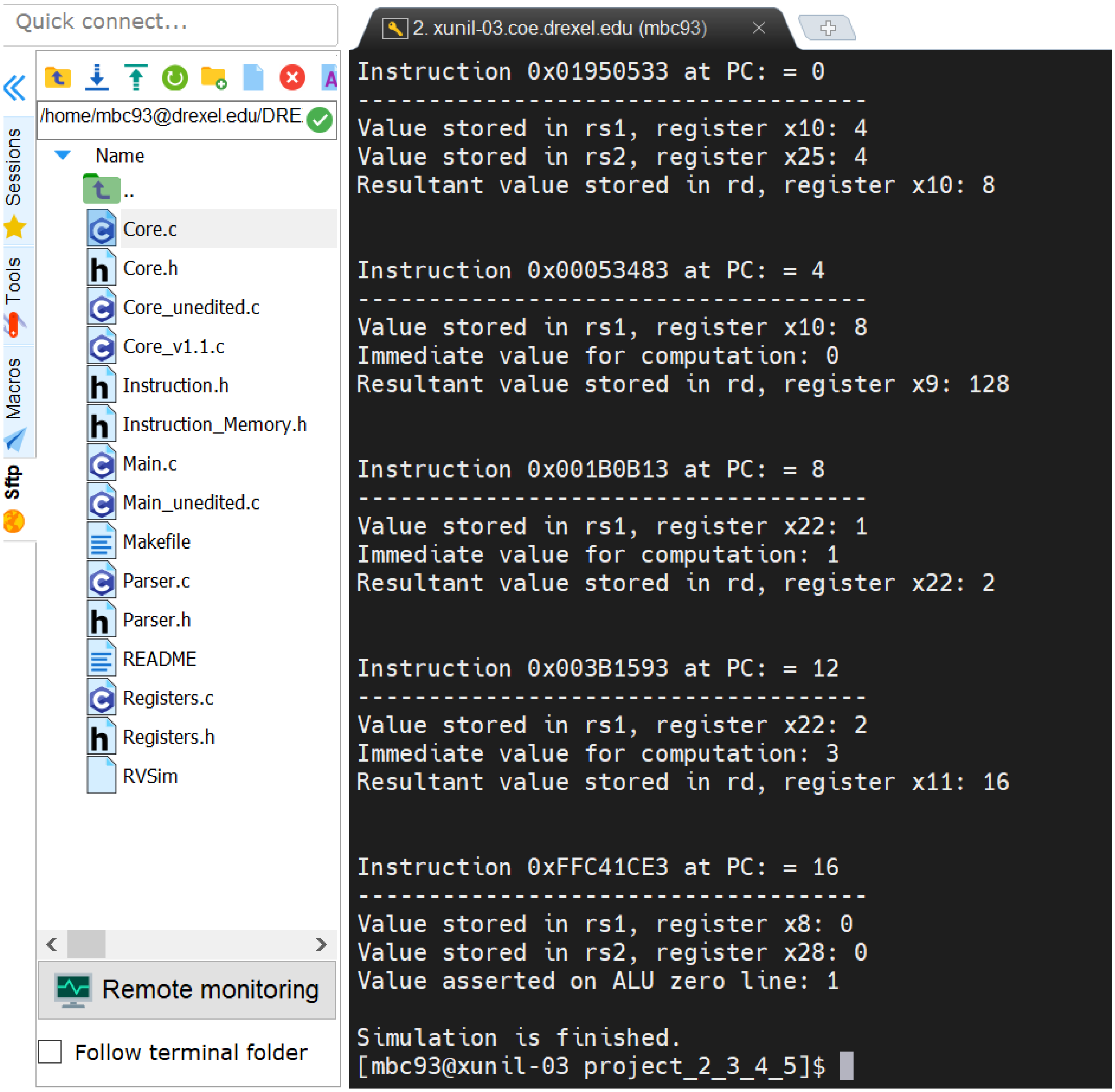
Project #2

Procedure Explanation:

1. **Assembly Instruction Evaluation, Byte-addressable Memory, and Little Endian Storage:** A three element array of unsigned 64-bit integers was defined and placed into a size-of-24-bytes data memory. Three assembly instructions were given and executed by hand using the given data memory. Answers to the two questions in part one of the assignment document and a visual representation of the size-of-24-bytes data memory containing the given array are presented on page 3 following the results of the simulation.

2. **Edit Core.{h,c} to Support a Single Cycle RISC-V Datapath:** Core.c for project_2, as given in the github ECEC355 directory, was extended to support the given assembly commands in the corresponding test file in the traces folder. The datapath was scripted such that it can support R-type, I-type, and SB-type instructions. The most challenging parts were decoding the instructions to pass the necessary values to generate the correct control signals for the different control units in the datapath and managing the control flow to achieve expected output. A screenshot of the generated output is given on page 2.

   Upon completion of the simulation the value stored in register x9 was 128 and the value stored in x11 was 16. These values can be seen in the screenshot on page 2.



```
Instruction 0x01950533 at PC: = 0
------------------------------------
Value stored in rs1, register x10: 4
Value stored in rs2, register x25: 4
Resultant value stored in rd, register x10: 8


Instruction 0x00053483 at PC: = 4
------------------------------------
Value stored in rs1, register x10: 8
Immediate value for computation: 0
Resultant value stored in rd, register x9: 128


Instruction 0x001B0B13 at PC: = 8
------------------------------------
Value stored in rs1, register x22: 1
Immediate value for computation: 1
Resultant value stored in rd, register x22: 2


Instruction 0x003B1593 at PC: = 12
------------------------------------
Value stored in rs1, register x22: 2
Immediate value for computation: 3
Resultant value stored in rd, register x11: 16


Instruction 0xFFC41CE3 at PC: = 16
------------------------------------
Value stored in rs1, register x8: 0
Value stored in rs2, register x28: 0
Value asserted on ALU zero line: 1

Simulation is finished.
[mbc93@xunil-03 project_2_3_4_5]$
```

```
ECEC 355 Project #2, Part #1: Written Portion/Theory
**********
* PART 1 *
**********
uint64_t arr[3] = {19088743, 2882400001, 169552957}
-------------------------------------------------------------
decimal         hex        [31:25]  [24:16]  [15:8]   [7:0]
-------------------------------------------------------------
19088743   = 0x01234567 = 00000001 00100011 01000101 01100111

2882400001 = 0xABCDEF01 = 10101011 11001101 11101111 00000001

169552957  = 0x0A1B2C3D = 00001010 00011011 00101100 00111101
-------------------------------------------------------------

a) In RISC-V an int occupies 4 bytes. However, when an int is loaded into data   memory
from storage memory it is loaded into a 64-bit register with it's MSB duplicated in bits
[63:32].

b) Data mapping for unsigned int arr[3] ~> assume base addr. of arr[] == MEM[0].
RISC-V ~> little endian - LSB sotred at lowest available data address.

MEM[0] -> MEM[23] represents a 24 BYTE little endian cache memory

            [7:0]     [15:8]    [24:16]    [31:25]    [39:32]    [47:40]    [55:48]    [63:56]
--------------------------------------------------------------------------------------------
MEM[0]    | 01100111 | 01000101 | 00100011 | 00000001 | 00000000 | 00000000 | 00000000 | 00000000 |
--------------------------------------------------------------------------------------------
MEM[0+8]  | 00000001 | 11101111 | 11001101 | 10101011 | 00000000 | 00000000 | 00000000 | 00000000 |
--------------------------------------------------------------------------------------------
MEM[0+16] | 00111101 | 00101100 | 00011011 | 00001010 | 00000000 | 00000000 | 00000000 | 00000000 |
--------------------------------------------------------------------------------------------

**********
* PART 2 *
**********
*** Assuming value stored in x23 == 0 ***

1. ld x8, 0(x23)
v[x8] == 00000000 00000000 00000000 00000000 00000001 00100011 01000101 01100111

2. ld x9, 16(x23)
v[x9] == 00000000 00000000 00000000 00000000 00001010 00011011 00101100 00111101

3. add x10, x8, x9
v[x10] = v[x8] + v[x9]
v[x10] = 19088743 + 169552957
v[x10] = 188641700 (decimal)
       = 00000000 00000000 00000000 00000000 00001011 00111110 01110001 10100100
       = 0x0B3E71A4
```

```
----------------------------------------------------------------------
ld x8 0(x23)

instruction type: I ~> imm[11:0] rs1 funct3 rd opcode
opcode: 0000011
rd: 8 ~> 01000
funct3: 011
rs1: 23 ~> 10111
immediate: 0 ~> 000000000000

machine instruction: 000000000000 10111 011 01000 0000011

----------------------------------------------------------------------
ld x9 16(x23)

instruction type: I ~> imm[11:0] rs1 funct3 rd opcode
opcode: 0000011
rd: 9 ~> 01001
funct3: 011
rs1: 23 ~> 10111
immediate: 16 ~> 000000010000

machine instruction: 000000010000 10111 011 01001 0000011

----------------------------------------------------------------------
add x10, x8, x9

instruction type: R ~> funct7 | rs2 | rs1 | funct3 | rd | opcode
opcode: 0110011
rd: 10 ~> 01010
funct3: 000
rs1: 8 ~> 01000
rs2: 9 ~> 01001
funct7: 0000000

machine instruction: 0000000 01001 01000 000 01010 0110011
```