CSC / CpE 142 Term Project Spring 2020

The objective of this project is to design and simulate the datapath and control unit for a pipelined system, which can execute the following assembly instruction set:

Function	syntax	opcode	op1	op2	funct. Code	type	Operation
Signed addition	add op1, op2	0000	reg	reg	1111	A	op1 = op1 + op2
Signed subtraction	sub op1, op2	0000	reg	reg	1110	A	op1 = op1 - op2
bitwise and	and op1, op2	0000	reg	reg	1101	A	op1 = op1 & op2
bitwise or	or op1, op2	0000	reg	reg	1100	A	op1 = op1 | op2
signed multiplication	mul op1, op2	0000	reg	reg	0001	A	op1 = op1 * op2 op1: Product (lower half) R0: Product (upper half)
signed division	div op1, op2	0000	reg	reg	0010	A	op1: 16-bit quotient R0: 16-bit remainder
Logical shift left	sll op1, op2	0000	reg	immd.	1010	A	shift op1 to the left by op2 bits
Logical shift right	slr op1, op2	0000	reg	immd.	1011	A	shift op1 to the right by op2 bits with sign extension
rotate left	rol op1, op2	0000	reg	immd.	1000	A	rotate left op1 by op2 bits
rotate right	ror op1, op2	0000	reg	immd.	1001	A	rotate right op1 by op2 bits
load	lw op1, immd (op2)	1000	reg	reg	N/A	B	op1 = Mem [immd + op2] (sign extend immd)
store	sw op1, immd (op2)	1011	reg	reg	N/A	B	Mem [immd + op2] = op1 (sign extend immd)
branch on less than	blt op1, op2	0100	reg	immd.	N/A	C	if (op1 < R0) then PC = PC + op2 (sign extend op2 & shift left)
branch on grater than	bgt op1, op2	0101	reg	immd.	N/A	C	if (op1 > R0) then PC = PC + op2 (sign extend op2 & shift left)
branch on equal	beq op1, op2	0110	reg	immd.	N/A	C	if (op1 = R0) then PC = PC + op2 (sign extend op2 & shift left)
jump	jmp op1	1100 set	off-	-------	N/A	D	pc = pc + op1 (sign extend op1& shift left)
halt	Halt	1111	-----	------	N/A	D	halt program execution

Each addressable memory location is a byte. The memory module can store 2^{15} words. Each instruction is 16-bit long (1 word). Each register is 16-bit long (1 word). There are 16 registers denoted by R0 through R15. Register R0 is used implicitly in branch instructions. There are 4 possible instruction formats:

Type A:

4- bit opcode	4-bit operand 1	4-bit operand 2	4-bit funct code

Type B:

4- bit opcode	4-bit operand 1	4-bit operand 2	4-bit offset

Type C:

4- bit opcode	4-bit operand 1	8-bit offset

Type D:

4- bit opcode	12-bit offset in jump -- unused in halt

You need to follow the design criteria discussed in Chapter 4 of the textbook. In particular, you must add enough logic to deal with hazards. Your system should also handle exceptions. In case of any exception (for example wrong opcodes, and an arithmetic overflow, etc.) you need to halt the execution of the program and display a message indicating the type of exception, that has occurred.

First, you need to design the datapath and control unit that meet the above requirements. Once the design is complete, model and simulate each component of the system in Verilog or Logisim. Verify the functionality of the individual components using stimulus files written in Verilog or a testing fixtures in Logisim. After modeling and validation of individual components, simulate the complete system in Verilog or Logisim. Use a machine-code program to test the simulated system. You must write your own assembly language for this. If I have time, I will provide you with some code and/or an assembler. You need to store the machine code version of the assembly program provided below in the memory unit before testing the design.

For Verilog, use a stimulus file to illustrate the functionality of the simulated system. In the stimulus file you must: i) instantiate the simulated system, ii) provide a clock generator block, iii) initialize all necessary parameters (such as reset) and provide any other necessary input, and iv) display input & output ports of major components such as PC, ADDERS, MEMORY, REGISTER FILE, ALU, and pipeline buffers (inputs and output) on every negative edge of the clock. These values will be used to test your system. It will also show the number of clock cycles required for each instruction. For Logisim you must make your project as visual as possible so that as many of these features can be directly read from displays while stepping through the simulation. Use good judgement here, the more effort that you put in, the better your grade.

The Verilog code you provide must the original work of the team. You must indicate if any Verilog code that you provide is not your original work and must list the reference(s) used to get the code. You only get credit for the code you designed
You must submit a typed report including the items listed below.

- *A Cover sheet (Include CSC/CpE 142, Term, Term Project, your name(s)*
- *Table of contents (with page number).*
- *Status report, what is completed, what isn't, etc.*
- *A diagram that shows the datapath. Properly label each component.*
- *The high representation of any control logic used in the design including the main control unit, ALU control unit, and any hazard detection, register forwarding used. (truth table or other forms)*
- *A hardcopy of the Verilog/Logisim source code and test file for each component of the datapath. If necessary, cut the computer printouts to 81/2 by 11 page size. Add sufficient comments to each file.*
- *The test assembly program used to test the simulated system with expected results.*
- *The stimulus module used for testing the system.*

In addition to the report you must submit the complete design electronically. You will be given instruction on how to submit electronic copy through email later.

You may work in groups of 2 for this project. It is your responsibility to find partners who are willing to work with you. Groups must work independently.

Test Assembly Code:
The register file and the memory module for your term project must be initialized based on the given immediate values or instructions. All immediate values and memory addresses are given as hexadecimal numbers. Registers are denoted by R0 - R15. In case of instructions, you need to First convert them into machine code and then store them in the memory. The memory and register file should be re-initialized with the given values on each reset.

Register	***Content***
======	======
R1	*0F00*
R2	*0050*
R3	*FF0F*
R4	*F0FF*
R5	*0040*
R6	*0024*
R7	*00FF*
R8	*AAAA*
R9	*0000*
R10	*0000*
R11	*0000*
R12	*FFFF*
R13	*0002*
Others 0000	

Address/Content

00	*ADD R1, R2*
02	*SUB R1, R2*
04	*OR R3, R4*
06	*AND R3, R2*
08	*MUL R5, R6*
0A	*DIV R1, R5*
0C	*SUB R0, R0*
0E	*SLL R4, 3*
10	*SLR R4, 2*
12	*ROR R6, 3*
14	*ROL R6, 2*
16	*BEQ R7, 4*
18	*ADD R11, R1*
1A	*BLT R7, 5*
1C	*ADD R11, R2*
1E	*BGT R7, 2*
20	*ADD R1, R1*
22	*ADD R1, R1*
24	*LW R8, 0(R9)*
26	*ADD R8, R8*
28	*SW R8, 2 (R9)*
2A	*LW R10, 2 (R9)*
2C	*ADD R12, R12*
2E	*SUB R13, R13*
30	*ADD R12, R13*
32	*EFFF*

Other memory locations = 0000

Data Memory

00	*2BCD*

Other memory locations = 0000