Alexandria University
Faculty of Engineering
Electrical Engineering Department

CS x35: Computer Architectures
Optional Design Project

Overview:
In this projects you will implement a subset of the multicycle MIPS architecture in HDL. You will implement a functioning outline of the multicycle processor for a small set of instructions, including: decoding all the instructions you will encounter in the projects, implementing most of the datapath, correct implementation of arithmetic and logic operations, and FSM control for these instructions.

Requirements:
Write HDL code for the multicycle MIPS processor. The processor should be compatible with the following top-level module. The mem module is used to hold both instructions and data. Test your processor using the testbench that will be given in this document.

```
module top(input logic clk, reset,
      output logic [31:0] writedata, adr,
      output logic memwrite);
      logic [31:0] readdata;
      // instantiate processor and memories
      mips mips(clk, reset, adr, writedata, memwrite, readdata);
      mem mem(clk, memwrite, adr, writedata, readdata);
endmodule

module mem(input logic clk, we,
      input logic [31:0] a, wd,
      output logic [31:0] rd);
      logic [31:0] RAM[63:0];
      initial
      begin
      $readmemh("memfile.dat", RAM);
      end
      assign rd = RAM[a[31:2]]; // word aligned
      always @(posedge clk)
      if (we)
      RAM[a[31:2]] <= wd;
endmodule
```

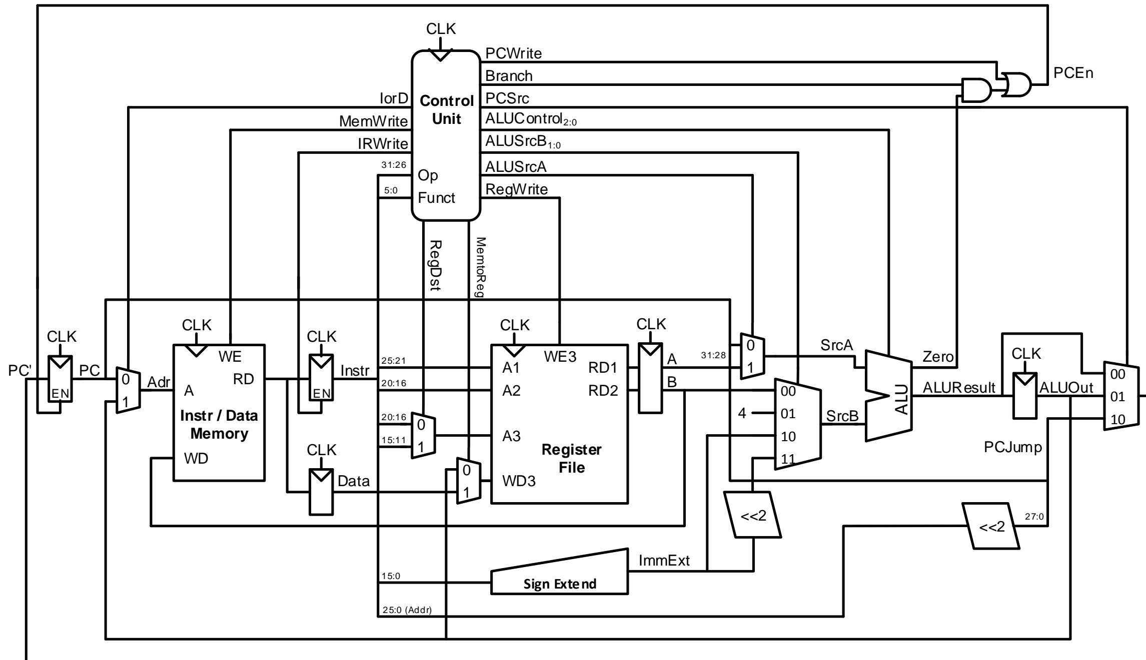
Your design should contain a program counter, a combined data and code memories, a register file, an ALU, and any other components needed, along with the instruction

decode and control circuits. Each instruction will be executed in an arbitrary number of clock cycles as needed by the instruction. Your processor must correctly execute all of the highlighted instructions in Table 1.

Table 1: MIPS Instruction Set

Opcodes	Example Assembly	Semantics
add	add $1, $2, $3	$1 = $2 + $3
sub	sub $1, $2, $3	$1 = $2 - $3
add immediate	addi $1, $2, 100	$1 = $2 + 100
add unsigned	addu $1, $2, $3	$1 = $2 + $3
subtract unsigned	subu $1, $2, $3	$1 = $2 - $3
add imm. Unsigned	addiu $1, $2, 100	$1 = $2 + 100
multiply	mult $2, $3	hi, lo = $2 * $3
multiply unsigned	multu $2, $3	hi, lo = $2 * $3
divide	div $2, $3	lo = $2/$3, hi = $2 mod $3
divide unsigned	divu $2, $3	lo = $2/$3, hi = $2 mod $3
move from hi	mfhi $1	$1 = hi
move from low	mflo $1	$1 = lo
and	and $1, $2, $3	$1 = $2 & $3
or	or $1, $2, $3	$1 = $2 | $3
and immediate	andi $1, $2, 100	$1 = $2 & 100
or immediate	ori $1, $2, 100	$1 = $2 | 100
shift left logical	sll $1, $2, 10	$1 = $2 << 10
shift right logical	srl $1, $2, 10	$1 = $2 >> 10
load word	lw $1, $2(100)	$1 = ReadMem32($2 + 100)
store word	sw $1, $2(100)	WriteMem32($2 + 100, $1)
load halfword	lh $1, $2(100)	$1 = SignExt(ReadMem16($2 + 100))
store halfword	sh $1, $2(100)	WriteMem16($2 + 100, $1)
load byte	lb $1, $2(100)	$1 = SignExt(ReadMem8($2 + 100))
store byte	sb $1, $2(100)	WriteMem8($2 + 100, $1)
load upper immediate	lui $1, 100	$1 = 100 << 16
branch on equal	beq $1, $2, Label	if ($1 == $2) goto Label
branch on not equal	bne $1, $2, Label	if ($1 != $2) goto Label
set on less than	slt $1, $2, $3	if ($2 < $3) $1 = 1 else $1 = 0
set on less than immediate	slti $1, $2, 100	if ($2 < 100) $1 = 1 else $1 = 0
set on less than unsigned	sltu $1, $2, $3	if ($2 < $3) $1 = 1 else $1 = 0
set on less than immediate	sltui $1, $2, 100	if ($2 < 100) $1 = 1 else $1 = 0
jump	j Label	goto Label
jump register	jr $31	goto $31
jump and link	jal Label	$31 = PC + 4; goto Label

Build your MIPS processor suing your preferred HDL language and you can use any component implemented in the course textbook. The multicycle MIPS architecture and main components you should build in your design are shown in Figure 1. You only need to modify the processor architecture to support the instructions highlighted in Table 1.



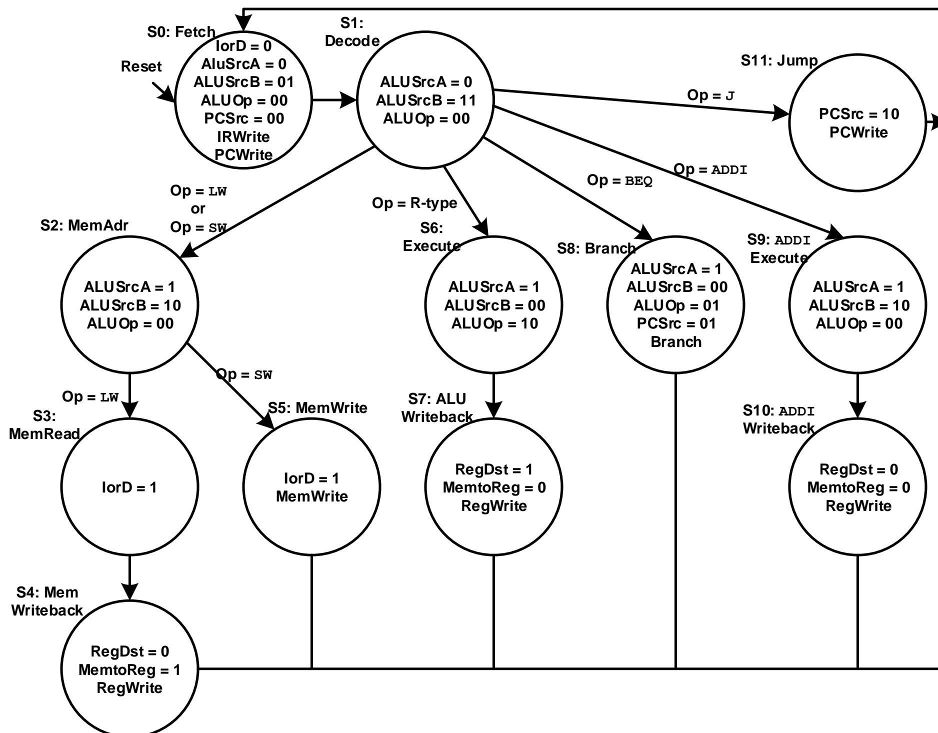


Figure 1: Multicycle MIPS Processor Architecture and controller's FSM

Testing

Write a test program in MIPS assembly that fully tests all of the features you have implemented. Our testing programs for this project will include a mixture of instructions from Table 1. This is a critical step, and you will use the MIPS testbench given by the

textbook and shown in this section. The MIPS testbench loads a program into the memories. The program in Figure 3 exercises some of the instructions by performing a computation that should produce the correct answer only if all of the instructions are functioning properly. Specifically, the program will write the value 7 to address 84 if it runs correctly, and is unlikely to do so if the hardware is buggy. This is an example of ad hoc testing. The machine code is stored in a hexadecimal file called memfile.dat, which is loaded by the testbench during simulation. The file consists of the machine code for the instructions, one instruction per line.

```
# mipstest.asm
# David_Harris@hmc.edu, Sarah_Harris@hmc.edu 31 March 2012
#
# Test the MIPS processor.
# add, sub, and, or, slt, addi, lw, sw, beq, j
# If successful, it should write the value 7 to address 84

#         Assembly              Description             Address   Machine
main:     addi $2, $0, 5        # initialize $2 = 5     0         20020005        20020005
          addi $3, $0, 12       # initialize $3 = 12    4         2003000c        2003000c
          addi $7, $3, −9       # initialize $7 = 3     8         2067fff7        2067fff7
          or   $4, $7, $2       # $4 = (3 OR 5) = 7     c         00e22025        00e22025
          and  $5, $3, $4       # $5 = (12 AND 7) = 4   10        00642824        00642824
          add  $5, $5, $4       # $5 = 4 + 7 = 11       14        00a42820        00a42820
          beq  $5, $7, end      # shouldn't be taken    18        10a7000a        10a7000a
          slt  $4, $3, $4       # $4 = 12 < 7 = 0       1c        0064202a        0064202a
          beq  $4, $0, around   # should be taken       20        10800001        10800001
          addi $5, $0, 0        # shouldn't happen      24        20050000        20050000
around:   slt  $4, $7, $2       # $4 = 3 < 5 = 1        28        00e2202a        00e2202a
          add  $7, $4, $5       # $7 = 1 + 11 = 12      2c        00853820        00853820
          sub  $7, $7, $2       # $7 = 12 − 5 = 7       30        00e23822        00e23822
          sw   $7, 68($3)       # [80] = 7              34        ac670044        ac670044
          lw   $2, 80($0)       # $2 = [80] = 7         38        8c020050        8c020050
          j    end              # should be taken       3c        08000011        08000011
          addi $2, $0, 1        # shouldn't happen      40        20020001        20020001
end:      sw   $2, 84($0)       # write mem[84] = 7     44        ac020054        ac020054
```

Figure 2: Assembly and machine code for MIPS test program

Documentation

The design document should include a block diagram showing all the major changes in the given architecture. You need not completely draw wires for control logic signals, but should indicate which components take control inputs, and give names to all control signals. Also include a description of your control and instruction decoding logic. For each control logic signal (or group of related control logic signals) you should provide (a) a brief description of what the signal does, e.g. what the values of the control signal mean; and (b) a truth table showing what value the signal takes for each possible opcode.