CS224
Section No.: 2
Spring 2018
Lab No.: 5
EFE ACER / 21602217 – TALHA MURATHAN GÖKTAŞ / 21601917

PRELIMINARY DESIGN REPORT

Question 1:

nop Instruction: IM[PC]
PC <- PC + 4

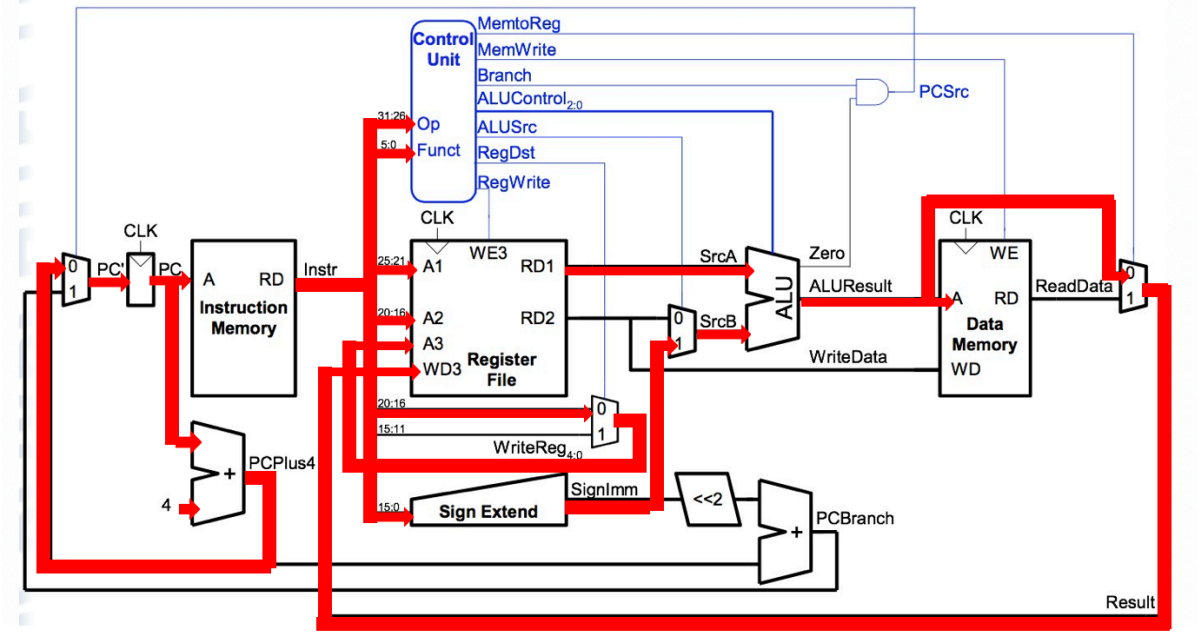
subi Instruction: IM[PC]
RF[rt] <- RF[rs] – SignExtImm
PC <- PC + 4

Question 2:

Changes for nop: There is no need for an additional hardware component to implement the nop instruction. The only requirement is to set the control signals such that addi $zero, $zero, 0 is executed. This way no change occurs in any content stored in the register file or the memory. The actual MIPS implements nop as sll $zero, $zero, 0; but in our case we chose to implement it as addi $zero, $zero, 0 to prevent the additional cost of the shamt wires and the logic to produce sll's ALUOp.

Changes for subi: There is also no need for an additional hardware component for the subi's implementation. Everything will be the same as addi's execution, except the fact that the Control Unit should produce 110 as the ALUControl signal to perform a subtraction.

The datapath is not changed for nop and subi and their data flow is the same as follows:



Question 3:

Main Control Table for the added instructions:

Instruction	*Opcode*	*RegWrite*	*RegDst*	*ALUSrc*	*Branch*	*MemWrite*	*MemToReg*	*ALUOp*	*Jump*
nop	001000	1	0	1	0	0	0	00	0
subi	001010	1	0	1	0	0	0	01	0

There is no need to change anything about the ALU Decoder Table. Since, both the nop instruction and the subi instruction follows the same datapath with addi in our design. We chose to set subi's opcode as 001010 as it does not exist for the core instruction set.

Question 4:

A sample MIPS assembly program to test the added instructions:

```
addi $v0, $zero, 0x0005
or $a0, $a3, $v0
and $a1, $v1, $a0
add $a1, $a1, $a0
beq $a1, $a3, 0x000A
slt $a0, $v1, $a0
sub $a3, $a3, $v0
sw $a3, 0x0044($v1)
lw $v0, 0x0050($zero)
nop
subi $a3, $a3, 0x0001
j 0x0000011
```

Question 5:

List of modules to be changed:

- maindec (opcode of subi should be decoded to produce the proper control signals)

```
////////////////////////////////////////////////////////////////////////////////////////////////

module maindec(input logic[5:0] opcode,
               output logic memToReg, memWrite, branch, ALUSrc, regDst, regWrite, jump,
               output logic[1:0] ALUOp);

logic [8:0] controls;

assign {regWrite, regDst, ALUSrc, branch, memWrite, memToReg, ALUOp, jump} = controls;

always_comb
  case(opcode)
    6'b000000: controls <= 9'b110000100; //R-type: 110000100
```

```
    6'b100011: controls <= 9'b101001000; //lw: 101001000
    6'b101011: controls <= 9'b0x101x000; //sw: 0x101x000
    6'b000100: controls <= 9'b0x010x010; //beq: 0x010x010
    6'b001000: controls <= 9'b101000000; //addi: 101000000
    6'b000010: controls <= 9'b0xxx0xxx1; //j: 0xxx0xxx1
    6'b001010: controls <= 9'b101000010; //subi: 101000010 (opcode: 001010) //CHANGE
    default: controls <= 9'bxxxxxxxxx; // illegal opcode
  endcase

endmodule
```

//

Note: The machine code for our nop instruction is:

001000 00000 00000 0000 0000 0000 0000
0010 0000 0000 0000 0000 0000 0000 0000
0x20000000

Hence, when someone writes nop as a symbolic machine instruction, it will be compiled to 0x20000000