ELEC6234
Embedded Processor Design

FPGA synthesis of a microprocessor core
Notes on ALU synthesis

picoMIPS block diagram - version 1

picoMIPS with branches and RAM

Data paths for arithmetic and logic instructions
operands and result are in registers, RAM (data memory) is not involved
Absolute jump: 
PC = branch address
where branch address is: Instruction[p-1:0]

Relative jump: 
PC = PC + Instruction[p-1:0]
Relative offset needs an adder within the PC

Absolute conditional branch: 
PC = [parallel] branch address: PC+1
where branch address is: Instruction[p-1:0]

Relative conditional branch: 
PC = [conditional] PC + Instruction[p-1:0]: (PC)+1

ALU is programmed for subtraction: Rs-Rs
Subtraction result is not stored but ALU Rflags Z, N, C are used to determine if the branch should be taken

Immediate jump
absolute or relative

Longer offset needs an adder within the PC

Decoder – version 1 (only ADD and SUB)

```verilog
// Author: tjk
// Function    : picoMIPS instruction decoder
// File Name   : decoder.sv

module decoder //combinational logic module
(
    input logic [5:0] opcode, // top 6 bits of instruction
    output logic [5:0] ALUfunc, // ALU function
    output logic [1:0] ALUflags, // ALU flags
    output logic [1:0] PCincr // PC increment
);

localparam IMM = 1'b0; // Immediate control
localparam ADD = 2'b00; // ADD instruction
localparam SUB = 2'b01; // SUB instruction
localparam NOP = 2'b11; // NOP instruction

ALUfunc = opcode[2:0]; // ALU function
PCincr = IMM; // PC increment

```
This decoder implements full PC control for conditional and unconditional branches.

```
begin
  // set default output signal values for NOP instruction
  PCincr = 1'b1; // PC increments by default
  PCabsbranch = 1'b0; PCrelbranch = 1'b0;
  ALUfunc = opcode[2:0];
  imm = 1'b0; w = 1'b0;
  takeBranch = 1'b0;
  case(opcode)
    `NOP: 
    `ADD, `SUB: begin // register-register
      w = 1'b1; // write result to dest register
      end
    `ADDI, `SUBI: begin // register-immediate
      w = 1'b1; // write result to dest register
      imm = 1'b1; // set ctrl signal for imm operand MUX
      end
    // relative conditional branches
    `BEQ: takeBranch = flags[1]; // branch if Z==1
    `BNE: takeBranch = ~flags[1]; // branch if Z==0
    `BGE: takeBranch = ~flags[2]; // branch if N==0
    `BLO: takeBranch = flags[0]; // branch if C==1
    default: $error("unimplemented opcode %h");
  endcase // opcode
end
```

This if statement implements the PC control logic for relative branches:

```
if(takeBranch) // relative branch condition is true;
  begin
    PCincr = 1'b0;
    PCrelbranch = 1'b1;
  end
end // always_comb
```

Notes:
1) syntax is simple: each line contains instruction code in hex followed by a comment
2) this is a simple test that uses NOP, LDI (Load Immediate to Register) and ADD instructions
3) LDI is a ‘synthetic’ instruction; LDI %d, imm is implemented as ADDI %d, %0, imm
Other instructions

- **J – Jump**
  - Syntax: $J<\text{address}>$
  - Operation: $P C = <\text{address}>$
- **JAL - Jump and link, simple subroutine call**
  - Syntax: $J A L<\text{subroutine address}>$
  - Operation: $P C = <\text{subroutine address}>, \%31 = PC+1$ (return address)
- **JR – Jump to address in a register**
  - Syntax: $J R\% r$
  - Operation: $P C = \% r$, (JR \% 31 may be used as return from subroutine)
- **MULT – multiply signed**
  - Syntax: $M U L T\% s, \% d$
  - Operation: $\% d = \% s * \% d$ (dedicated registers might be needed in hardware)

RAM instructions

- **LW – load word from RAM**
  - Syntax $L W\% d, \% s, \text{imm}$
  - Operation: $\% s = RAM(\% d + \text{imm})$
- **SW – store word in RAM**
  - Syntax $S W\% d, \% s, \text{imm}$
  - Operation: $\text{RAM}(\% d + \text{imm}) = \% s$
- **Load must be performed in two clock cycles if the RAM is synchronous**
  - This can be implemented as two instructions in the program memory
    - *ADD – address register, program ALU and clock RAM*
    - *ADD – address RAM output data register*

Summary

- **Data path**
  - ALU, General Purpose Registers,
- **Control path**
  - Program Counter, Program Memory,
  - Simple decoder with essential instructions
    - ADD, SUB, ADDI, SUBI, BEQ, BNE, BGE, BLG
- **CPU encapsulating module to allow integration of modules developed so far and execution of simple programs**
- **Major features still to be implemented:**
  - RAM, load and store instructions,
  - subroutine calls and returns
  - multiplier
  - shifter