# nitcrisc-24

## a reduced instruction set processor

computer architecture and design
monsoon 2024

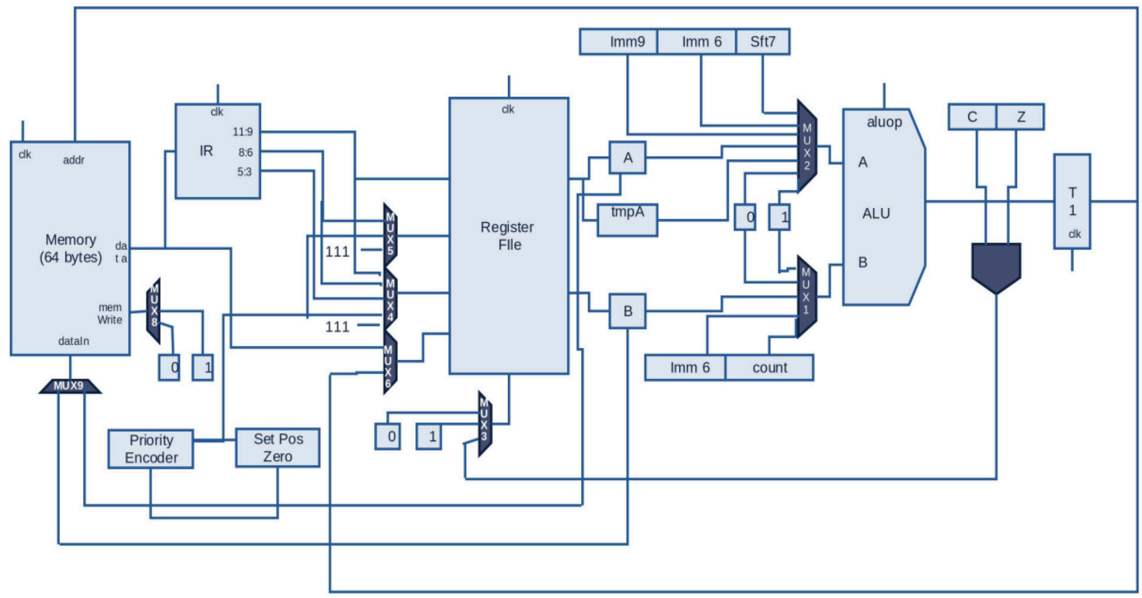
assignment 1
oct 06 2024

naila fathima
b220048cs
arun natarajan
b220019cs

# Objective:

To develop a 16-bit multicycle processor, NITC-RISC18. It is an 8-register, 16-bit computer system and uses registers R0-R7 for general purposes and a Condition Code register to store carry abd zero flags.  Register R0 always stores the program counter. Both instruction memory and data memory are word-addressable and it supports the following three types of instructions.

# Datapath:




# Modules and their Uses:

| Module | Purpose |
|---|---|
| test1 | Top level module that instantiates processors and memory |
| mips | Sets the instruction flow - instantiates controller and datapath |
| mem | The inputs are clock signal, write enable, a 16 bit write address, 16 bit data to be written, (if write is enabled and its positive edge of clock). A register array RAM is initialised which has 64 locations and each location stores a 16 bit data. The memory is preloaded with hexadecimal |

| | |
|---|---|
| | values from memfile.dat using the $readmemh system call in the beginning of every simulation. Read is an asynchronous operation whereas Write is a synchronous operation. |
| Controller ( the heart of multicycle :) | Inputs are the clock, reset, compare signals and a 4-bit opcode fetched from the instruction. Outputs will be a pc-write-enable (pcen), memory-write (memwrite), instruction-register-write(irwrite), register-write(regwrite) signals, alusrca, iord, memtoreg, regdst, alusrcb, pcsrc (2-bits for more options), alucontrol. Instatntiates the main decoder and the aludecoder, pcen is set to 1 if either pcwrite is 1 or both the branch and compare are 1. |
| maindec | Sets all the states for the FSM and the control signals accordingly. |
| aludec | Maps the 4-bit opcodes to 3-bit alu codes for performing desired operations. |
| datapath | Fetches instructions from memory, decodes them, and performs operations using the ALU. Interacts with the register file and memory to read/write data. Updates PC based on control signals, allowing sequential execution or branching. ALU performs arithmetic and logical operations as desired. |
| regfile | Represents a register file that contains 8 registers each capable of storing 16 bits. Inputs are the clock signal, IR_CZ - a 2 bit signal that controls writes to registers depending on carry and zero flags, 3 bit alucode, write enable signal, read addresses and write addresses, write data to be fetched. Outputs the the data being read. |
| sl1 | Shifts the 16-bit input by 1 position for calculating the branch addresses. |
| flopenr | A parameterised width (default = 8 bits) flipflop that updates the value if enable is set to 1 and updates the value to 0 if reset = 1 at the posedge of clock and posedge of reset. |
| flopr | A parameterised width (default = 8 bits) flipflop that updates the value to 0 if reset = 1 otherwise updates the value to the input passed at the posedge of clock and posedge of reset. |
| signext | Sign extends the input to 16 bits by replicating the msb. |

| | |
|---|---|
| mux2 | A parametrised width 2x1 multiplexor |
| mux3 | A parametrised width 3x1 multiplexor |
| mux4 | A parametrised width 4x1 multiplexor |
| carry_generate | Calculates if there is a carry bit generated due to the ALU operation |
| alu | Performs addition, nand, comparison depending on the aluOp code. |

# States and their Descriptions:

| **States** | **Code** | **Description** |
|---|---|---|
| FETCH | 00000 | The processor fetches the instruction from memory. The instruction pointed by the PC is read into the Instruction Register (IR) |
| DECODE | 00001 | Decode the fetched instruction to understand what kind of operation it is and determine the next steps. |
| MEMADR | 00010 | Calculate the memory address for load/store instructions. |
| MEMRD | 00011 | Read data from memory for a load instruction. |
| MEMWR | 00100 | Write data to memory for a store instruction. |
| EXECUTE | 00101 | Perform an ALU operation (Addition, Comparison or NAND). |
| ALUWRITEBACK | 00110 | Write the result of an ALU operation back to a register. |
| BRANCH | 00111 | Used to compute the target address for a branch and update the PC if the branch condition is met. |
| JALRW | 01000 | Used to store the return address of JAL. |
| JALPC | 01001 | Used to update the PC to target address. |

# Flow of Different Instructions:

***Instruction Encoding*:**

| | | | | | | |
|---|---|---|---|---|---|---|
| ADD: | 0000 | RA | RB | RC | 0 | 00 |
| ADC: | 0000 | RA | RB | RC | 0 | 10 |
| NDU: | 0010 | RA | RB | RC | 0 | 00 |
| NDZ: | 0010 | RA | RB | RC | 0 | 01 |
| LW: | 1010 | RA | RB | 6-bit Immediate | | |
| SW: | 1001 | RA | RB | 6-bit Immediate | | |
| BEQ: | 1011 | RA | RB | 6-bit Immediate | | |
| JAL: | 1101 | RA | 9-bit Immediate | | | |

** RA: Register A, RB: Register B, RC: Register C*
** All immediate values are signed*

***Instruction Description*:**

| **Name** | **Type** | **Assembly** | **Action** |
|---|---|---|---|
| ADD | R | add rc, ra, rb | ADD the contents of ra and the contents of rb and store the result in rc.<br><br>It modifies the carry flag and zero flag. |
| ADC | R | adc rc, ra, rb | ADD the contents of ra and the contents of rb and store the result in rc if the carry flag is set.<br><br>It modifies the carry flag and zero flag. |
| NDU | R | ndu rc, ra, rb | NAND the contents of ra and the contents of rb and store the result in rc.<br><br>It modifies the zero flag. |
| NDZ | R | ndz rc, ra, rb | NAND the contents of ra and the contents of rb and store the result in rc if the zero flag is set.<br><br>It modifies the zero flag. |
| LW | I | lw ra, rb, imm | Load value from data memory into ra. The memory address is formed by adding the contents of rb with immediate 6 bits.<br><br>It modifies the zero flag. |
| SW | I | sw ra, rb, imm | Store the contents of ra into data memory. The memory address is formed by adding the contents of rb with immediate 6 bits. |
| BEQ | I | beq ra, rb, imm | If the contents of ra and rb are the same, then branch to (PC+imm), where PC is the address of the beq inst. |
| JAL | J | jal ra, imm | Branch to (PC+imm)<br>Store (PC+1) into ra, where PC is the address of jal inst. |

** RA: Register A, RB: Register B, RC: Register C*
** All immediate values are signed*

## ADD :

| | | | | |
|---|---|---|---|---|
| Instruction gets stored in IR | Values of reg read into A and B for Add | Added answer written into Rc | Read PC into B and +1 passed to ALU | PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle) |

## ADC :

| | | | | | |
|---|---|---|---|---|---|
| Instruction gets stored in IR | Values of reg read into A and B for Add | Logical combination of the carry and zero bits is stored in CZ_reg for Add | Depending upon logical combination of the carry and zero bits added value is written in Rc | Read PC into B and +1 passed to ALU | PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle) |

## NDU:

| | | | | |
|---|---|---|---|---|
| Instruction gets stored in IR | Values of reg read into A and B for NAND | NANDed answer written into Rc | Read PC into B and +1 passed to ALU | PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle) |

## NDZ:

| | | | | | |
|---|---|---|---|---|---|
| Instruction gets stored in IR | Values of reg read into A and B for NAND | Logical combination of the carry and zero bits is stored in CZ_reg for NAND | Depending upon logical combination of the carry and zero bits added value is written in Rc | Read PC into B and +1 passed to ALU | PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle) |

## LW:

| | | | | |
|---|---|---|---|---|
| Instruction gets stored in IR | Imm 6 and reg B is passed to ALU | Data output from memory is stored in the Register File | Read PC into B and +1 passed to ALU | PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle) |

## SW:

| | | | | | |
|---|---|---|---|---|---|
| Instruction gets stored in IR | Imm 6 and reg B is passed to ALU | Resultant memory address, is passed to memory block | Desired data is written in the memory | Read PC into B and +1 passed to ALU | PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle) |

## BEQ:

| | | | | | |
|---|---|---|---|---|---|
| Instruction gets stored in IR | reg A and reg B values are loaded | Same as previous state to ensure proper data storage in T1 register | Imm 6 and reg B is passed to ALU | Read PC into B and +1 passed to ALU | PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle) |

## JAL:

| | | | | | |
|---|---|---|---|---|---|
| Instruction gets stored in IR | Read PC into B and +1 passed to ALU | Store PC+1 in reg A | add imm6 with PC, caring not to set the carry | Store the obtained result in PC | PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle) |