Mips Processor

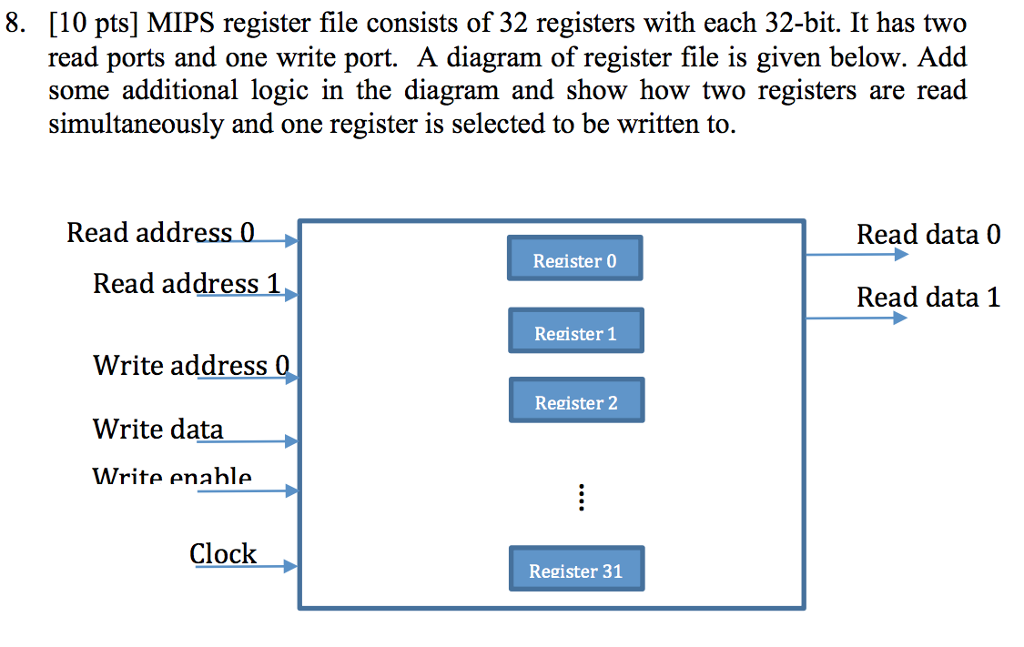
A simple 32-bit Mips processor that support the main operations and support pipelining.

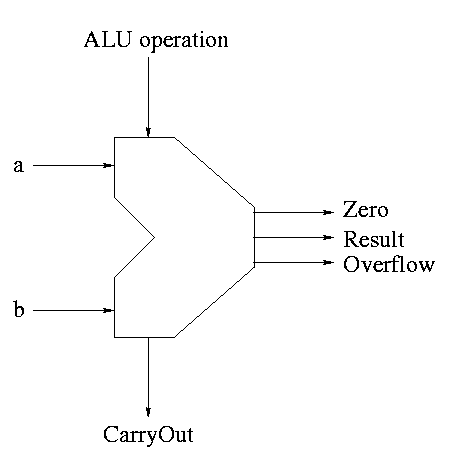
List of supported operations:

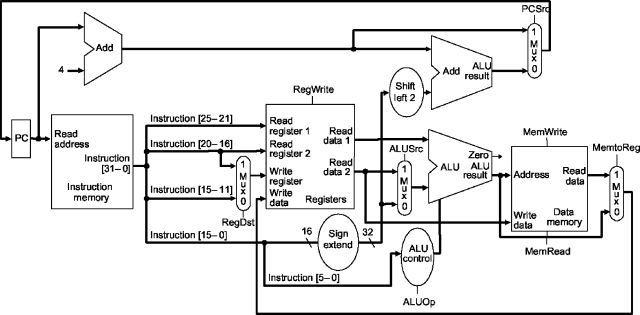
1. All R-Type (add, sub, jr, and, or ,…)
2. All I-Type (addi, andi, lw, sw, ..)
3. Jump operation

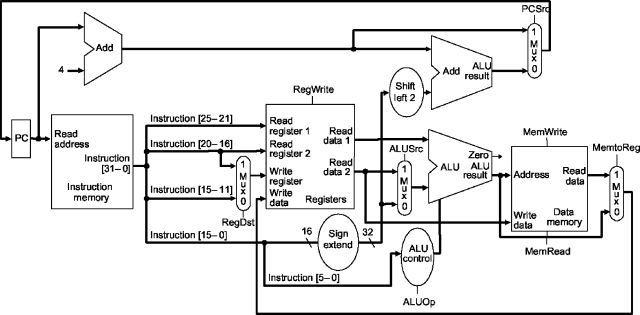
**Detailed description of our processor Modules:**

***PC module:*** here we have our instruction file, we read from it line by line and execute the instructions on it.

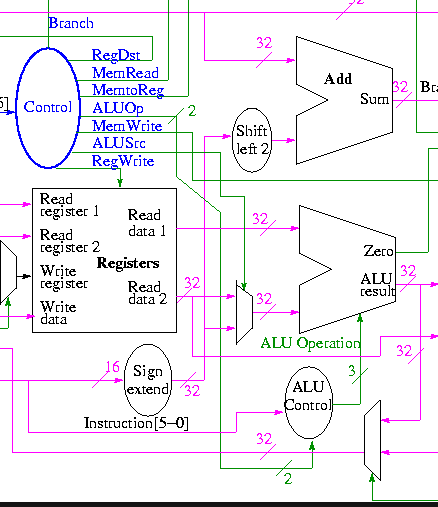
 ***Register File:*** 32 registers used to save data and read from it

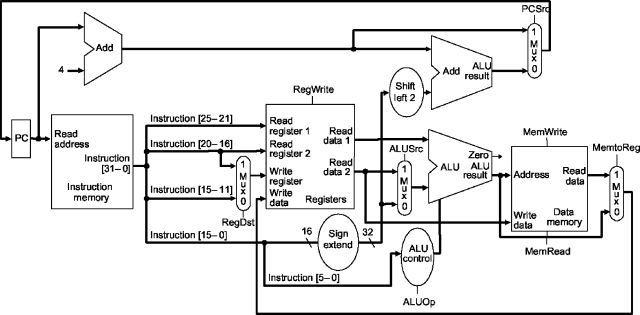
 ***ALU:***  takes two inputs and a control signal and output the operation result and zero flag to be used in beq and overflow indicator.

 ***Data memory:*** a 1024 memory that support read and write operations

 ***Sign Extend:*** takes a binary value and extend it based on its sign.

***Control Unit:*** takes the op code from the instruction and send the required control signals.



 ***Alu Control:*** takes the ALUOP and send the operation signal to the ALU.

**Control Hazards**

• Control hazards occur when we don’t know which instruction to execute next

• Mostly caused by branches

• Strategies for dealing with them:

• Stall

• Guess!

• Leads to speculation

• Flushing the pipeline

• Strategies for making better guesses

**Reason I’ve chosen those test\_cases:**   
- To test any control hazards which is mainly caused by “beq” and “bne” instructions

**Test Cases:**

**1) so.txt:** a program to simple operations without hazards.

*\*C code:*

- f = (g + h) - (I + j);

- e = f;

\* T*ested operations:*

*-* add

- sub

- sw

\* E*xpected output:*

*PC= 0 instruction=add alu result=3 writeData= 3*

*\* Real output:*

**2) lwhazard.txt:** a program to test data hazards after load word.

*\*C code:*

- x = x + 5;

\* T*ested operations:*

*-* addi

- lw

- sw

\* E*xpected output:*

*\* Real output:*

**3) beq.txt:** a program to test both data and control hazards.

*\*C code:*

- x = x + 5;

\* T*ested operations:*

*-* addi

- lw

- sw

- slt

- beq

- j

- add

\* E*xpected output:*

*\* Real output:*

**4)test\_case:** **test1.txt** a program to more complicated operations without hazards.

**\*C code:**

if (i<N) {

A[i] = 0;

}

**\*Tested operations:**

- lw

- sw

- beq

- sll

- add

**\*Expected output**

**\*Real output**

**5) test\_case:** **ch.txt** a program to more complicated operations without hazards.

**\*C code:**

for(i=0; i < n; i++) {

for(j=0; j < m; j++) {

sum += a[i][j]

}

}

**\*Tested operations:**

- add

- addi

- bne

**\*Expected output**

**\*Real output**

**6)test\_case:** **ch2.txt** a program to more complicated operations without hazards.

**\*C code:**

for (i=0; i<N; i++) {

A[i] = MAX\_SIZE;

}

**\*Tested operations:**

- add

- lw

- sll

- ori

- slt

- beq

- sw

- addi

- j

**\*Expected output**

**\*Real output**

**How To Run:**

- open terminal and run $python assym.py

- then write the instruction file name.

- you will get the machine code file of this instructions to use it in the verilog file

- then run the verilog file.