**明文 0416329**

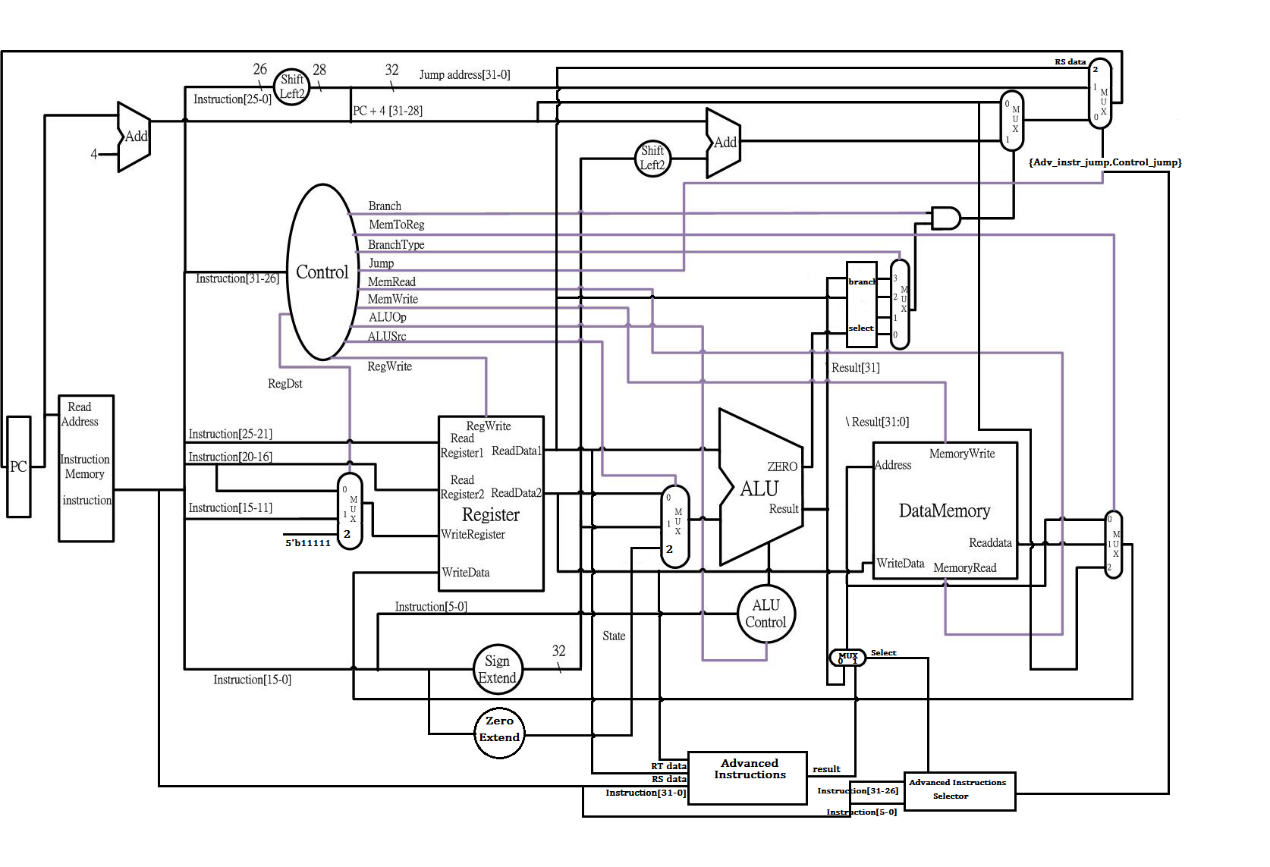
**文合西 0416330**

**Computer Organization**

**HDL simulator:**

Vivado

**Architecture diagram:**



**Detailed description of the implementation:**

The main modules of the Single Cycle CPU are implemented the same way as the ones in the previous lab (lab 2). Here are the new or modified modules:

**Zero\_Extend:**

Its function is similar to the original Sign extend. This module extends the immediate field with 16 0’s. Its output is later used to implement the LUI (load unsigned immediate) function.

**MUX\_ALUSrc (modified):**

Extended to be a 3-to-1 MUX. The third input field is the zero\_extend’s output. Its output then goes to the ALU module.

**MUX\_write\_reg (modified):**

Extended to be a 3-to-1 MUX. The third input field is a fixed 5’b11111, representing the 31st register. This input is selected when the JAL instruction is executed, in order to write to the 31st register.

**Decoder (modified):**

The decoder adds Branch\_o, Jump\_o, MemRead\_o, MemWrite\_o, MemtoReg\_o ,Branchslt\_o, Jumpal\_o as extra outputs, in order to enable the instructions required for this lab.

**Data\_Memory:**

Holds all the information stored in memory. It also has the capabilities of retrieving and writing data to/from memory. If data is retrieved from memory, it may be written further to a selected register.

**Memory\_to\_reg:**

It has 3 input fields; the first comes from the Data Memory’s Read data output field, the second one comes from the resulting selection between the advanced\_instructions module (implemented in lab 2) and the ALU. The third one is based on our implementation: It takes the PC+4 value as the input field, so it can be written to the register[31] when the JAL instruction is executed.

**Branch\_select (As a substitute to the branch selector MUX):**

In our implementation, we reuse the branch\_select module used in lab 2. It takes as inputs the result and zero provided by the ALU, and the RS register’s most significant bit. We select to branch-on-equal, branch-on-greater-or-equal-to-zero, branch-on-greater-than or branch-on-not-equal-to-zero.

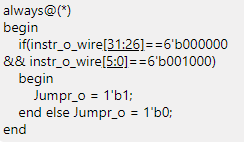
**Adv\_instr\_sel (modified):**

We added an extra output (adv\_instr\_jump). This output is set to one when the JR instruction is called. This output is then concatenated with the decoder’s jump output.

**Mux\_jump:**

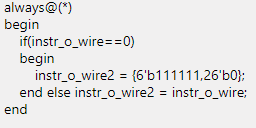
Three inputs: First input is the result of the selecting between branching or not, second input is the jumping address, third input is the RS register data (used when the JR instruction is executing). The select input is 2’b long, and is a concatenation of the adv\_instr\_jump output and the decoder’s jump output.

**To implement the jumps we use the following code:**



**Problems encountered and solutions:**

We were having trouble implementing the no-operation and at the end our solution as to implement this always loop on our top module. Besides that there were no major problems, just a matter of implementing the new signlas and changing the correct inputs to our modified and new modules.



**Lesson learnt (if any):**

On this lab we earned experience on working from a preexisting project and expanding it to add more functionality to it. This preexisting project being our lab 2 and the expanded functionality the tasks that were asked on this lab. Also having to translate the MIPS instructions into machine code gave us a better insight and understanding at how our fetches the instructions and works with them.