

# **Scalar Pipelined MIPS CPU Architecture**

**Laboratory five assignment**

**Hanan Ribo**

**05.06.25**

## Table of contents

1.	Aim of the Assignment.....	3
2.	Definition and prior knowledge.....	3
3.	Assignment definition.....	3
4.	Single cycle and Pipelined MIPS architecture diagrams.....	4
5.	Pipelined MIPS architecture requirements .....	5
6.	System design flow.....	7
7.	Compiler, Simulator and Memory.....	9
8.	CPU core design verification and QA test.....	9
9.	Requirements .....	10
10.	Grading policy .....	11
11.	References.....	11

## 1. Aim of the Assignment

- Design, synthesis and analysis of a simple MIPS compatible CPU.
- Scalar Pipelined MIPS CPU Architecture (from single cycle MIPS CPU Architecture)
- Understanding of CPU vs. MCU concept.
- Understanding in FPGA memory structure.

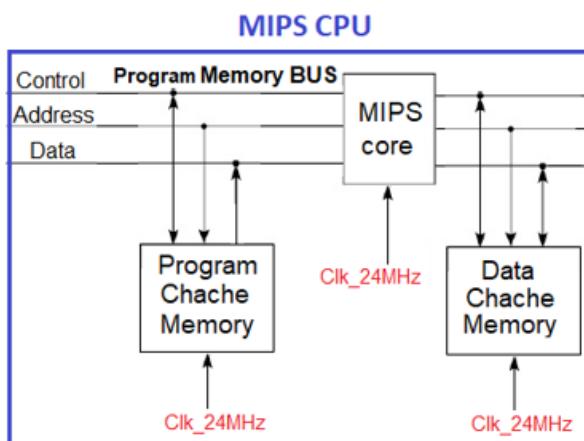
## 2. Definition and prior knowledge

The aim of this laboratory is to design a Scalar Pipelined MIPS CPU from a single cycle MIPS compatible CPU. The CPU core must be capable of performing instructions from a given MIPS instruction set. The design will be executed on the Altera Board. The MIPS architecture is Harvard architecture in order to increase throughput and simplify the logic. **You are required to support the Assembly Instruction Set given in the dedicated attached file.** For additional information regarding MIPS CPU, Architecture, ISA and instructions see MIPS technical documents [1].

## 3. Assignment definition

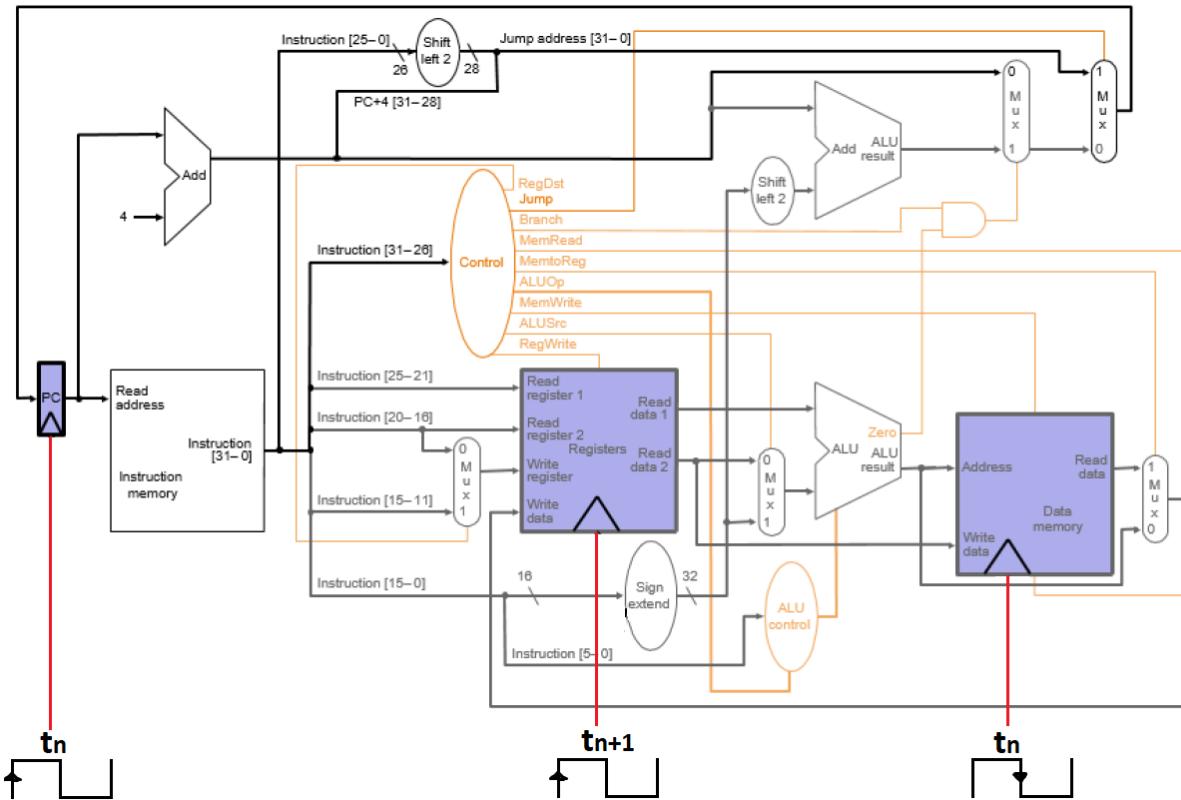
The architecture must include a MIPS ISA compatible CPU with data and program memory L1 Caches for hosting data and code. The block diagram of the architecture is given in Figure 1. The CPU will have a standard MIPS register file. The top level and the MIPS core must be structural. The design must be compiled and loaded to the Altera board for testing. A single clock (CLK) should be used in the design.

**Note:** use push-button **KEY0** as a core RESET (brings PC to the first program command).

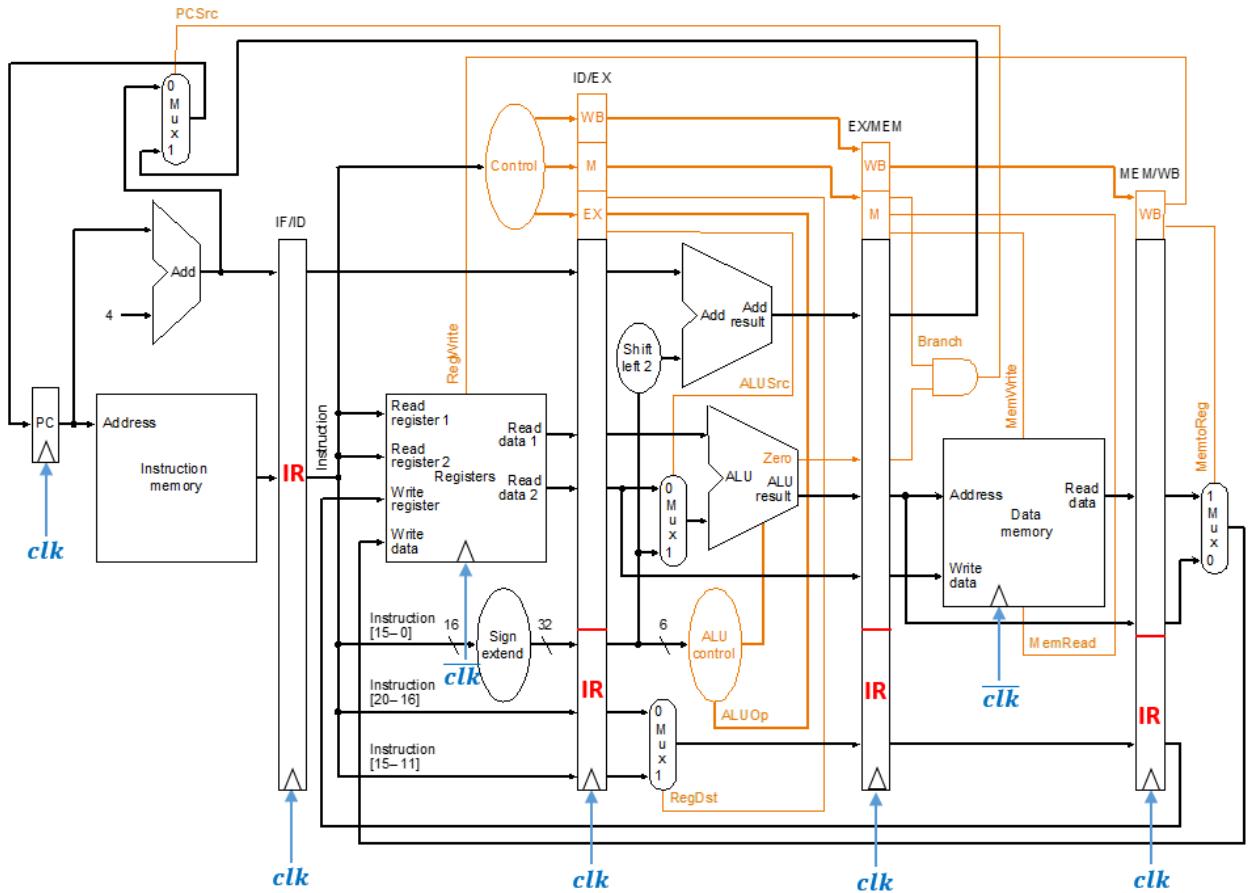


#### 4. Single-cycle and Pipelined MIPS architecture diagrams

## Single cycle MIPS diagram – Figure 2:



### Pipelined MIPS diagram (without interlocking) – Figure 3:



## 5. Pipelined MIPS architecture requirements

The single-cycle MIPS architecture you have given needs to be elevated to a pipelined MIPS architecture, shown in Figure 6. The development process is going through intermediate phases, shown in Figures 4 and 5. Figure 4 shows a pipelined MIPS architecture with Stall detection Logic based on a Combinational Check approach as a Data Hazard detection unit. Figure 5 shows a pipelined MIPS architecture with a Data Hazard detection unit and a Data Forwarding Unit when branches are resolved in the pipeline stage 4. Figure 6 (the required design) illustrates a pipelined MIPS architecture featuring a Data Hazard detection unit and a Data Forwarding Unit, where branches are resolved in a single delay slot.

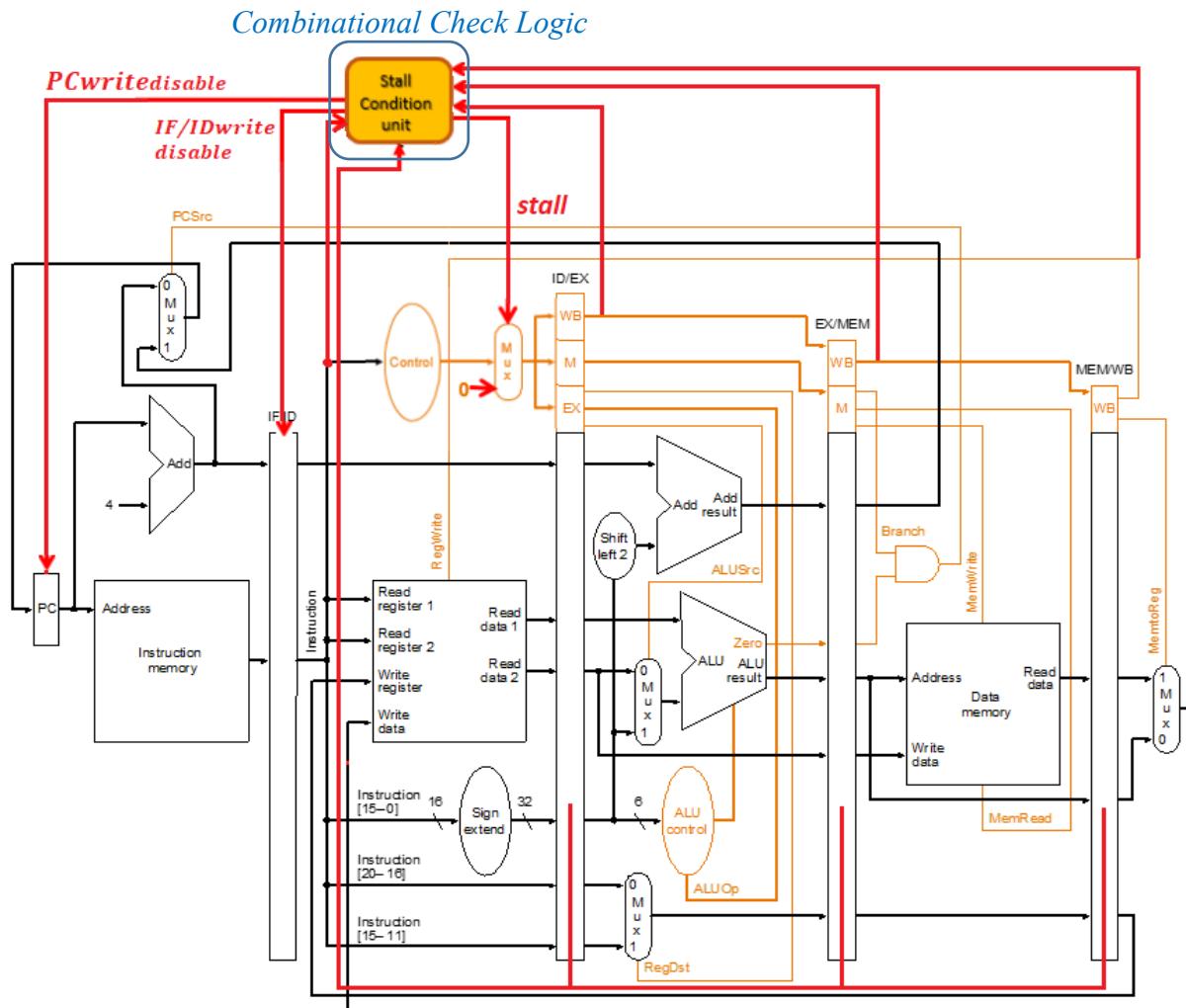


Figure 4: five stages pipelined MIPS architecture without forwarding support

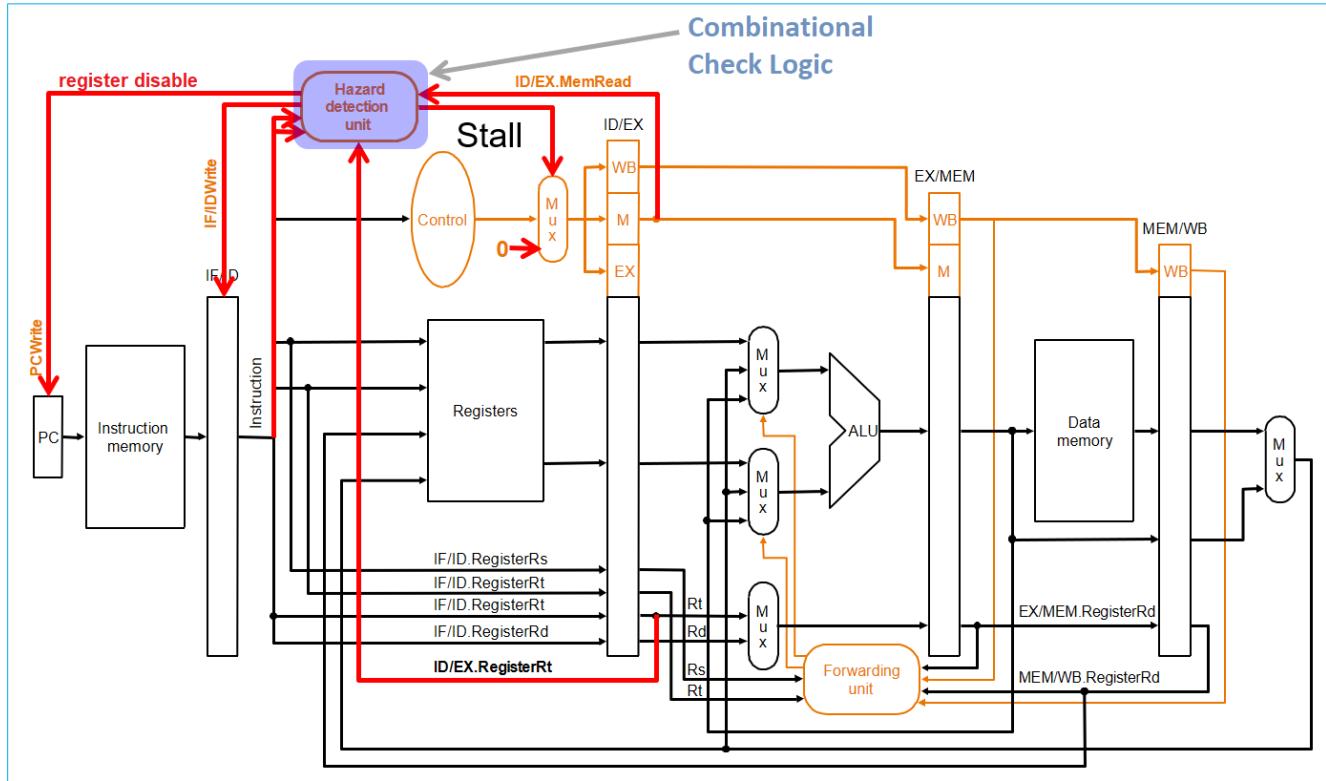


Figure 5: five stages pipelined MIPS architecture with forwarding support

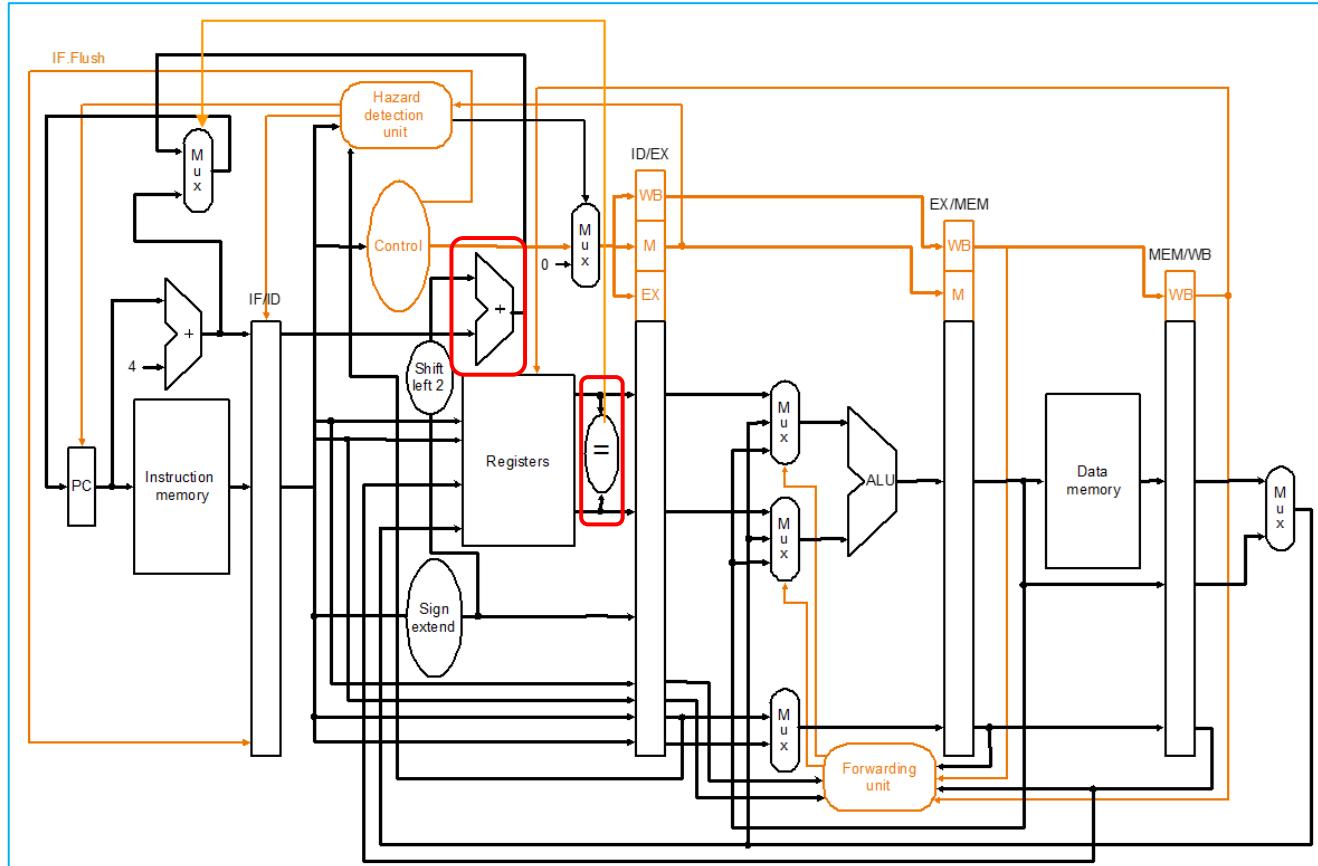
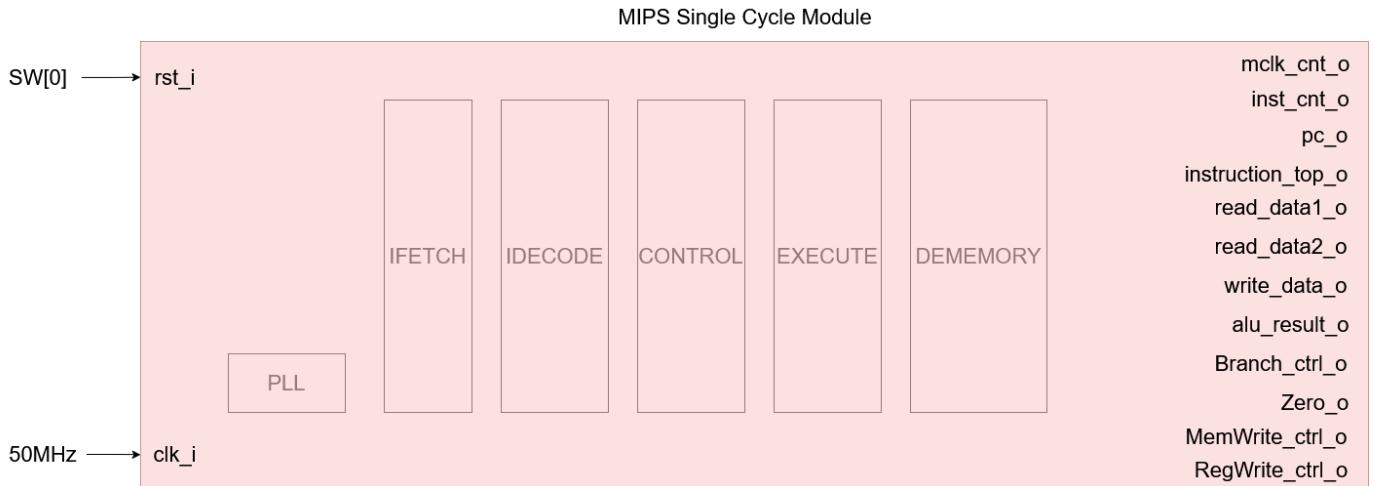


Figure 6: Five-stage pipelined MIPS architecture with forwarding and single delay slot support

## 6. System design flow

The design development flow is contained from functional verification using ModelSim (start phase) with the ability to discern each line in the design and Signal Tap (electrical) verification (second phase) with limited ability to discern only few lines. In order discern important lines in pipelined MIPS CPU using signal tap, there is need to add desirable lines shown in Figure 8 to the top entity (in comparison to single cycle top entity you were given in preparation material – figure 7).



**Figure 7: Single-cycle MIPS architecture top entity**

In addition to the desirable lines to be added to the pipelined MIPS top-level entity, you are required to add the following essential registers to support IPC calculation and breakpoint debug ability.

- i. **STCNT\_o** (stall counter register) 8-bit register – **STCNT\_o** cleared on reset:

This register is a core stalls counter that counts on the clock rising edge when a stall occurs.

- ii. **FHCNT\_o** (flush counter register) 8-bit register – **FHCNT\_o** cleared on reset:

This register is a core flush counter that counts on the clock rising edge when a flush occurs.

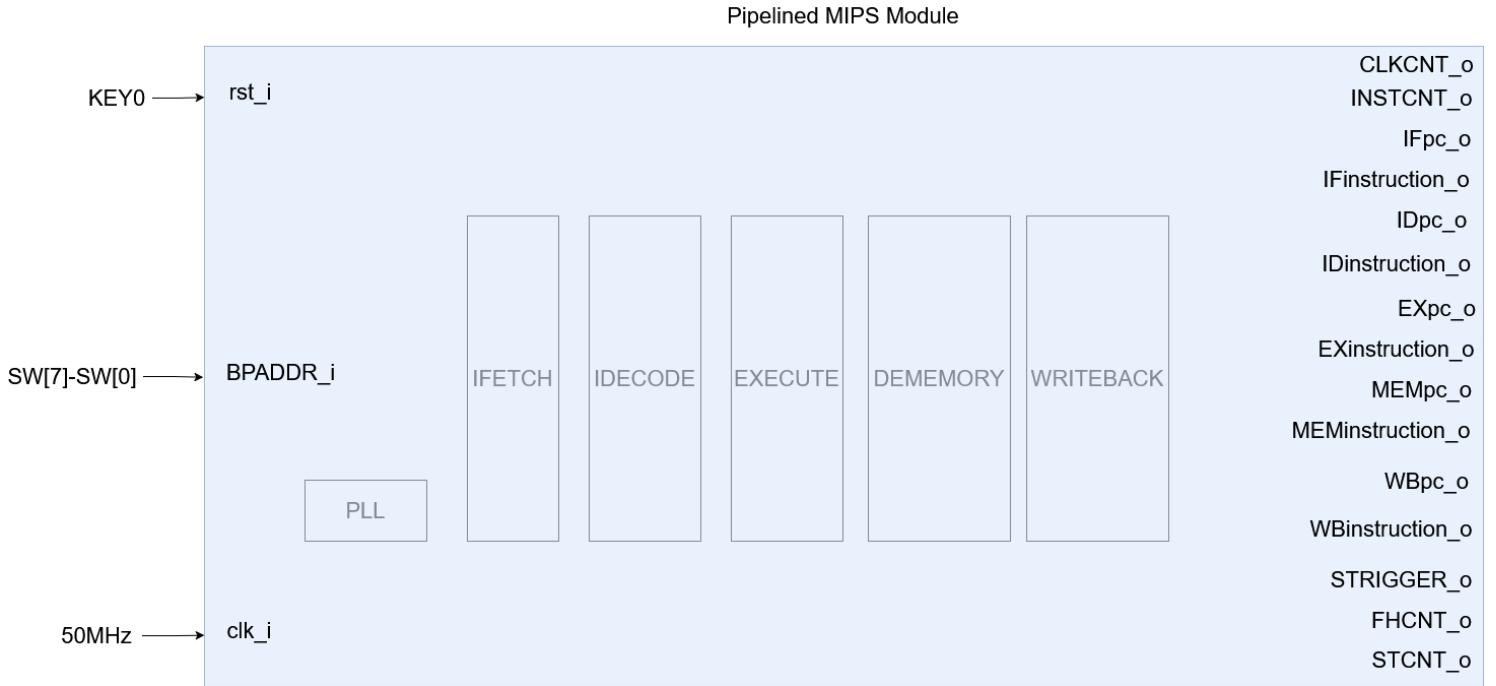
- iii. **BPADDR\_i** (break point address) 8-bit register:

This register uses a breakpoint address in order to feed logic that issues a signal tap trigger event. This signal tap trigger enables us to track CPU core lines from any PC address until the signal tap buffer is full. **BPADDR\_i** is cleared on reset. *Using the Auto-run button in the signal tap window, we can debug the CPU core as much as we wish in only one signal tap session.* **BPADDR\_i** register is fed by SW7-SW0 located on the FPGA board. The Signal tap trigger equation is:

$$\text{SignalTaptrigger} = (\text{PC} == \text{BPADDR}_i)$$

**Performance:  $\text{IPC} = \frac{\text{CLKCNT}_o - (\text{STCNT}_o + 4 + \text{dept} * \text{FHCNT}_o)}{\text{CLKCNT}_o} = \frac{\text{INSTCNT}_o - \text{dept} * \text{FHCNT}_o}{\text{CLKCNT}_o} = \frac{\text{MARSInstructionCounter}}{\text{CLKCNT}_o}$**

where:  $\text{IPC} = 1/\text{CPI}$



**Figure 8: Pipelined MIPS architecture top entity**

**Note:** The capability of Signal Tap verification in terms of a number of channels versus channel dept is given under the total memory condition equation:

$$STdept = \frac{\text{Embedded memory size} - \text{Design usage memory size}}{\#STchannels}$$

- The CPU will be based the *standard 32bit MIPS ISA* and the Instructions will be 32 bit wide. The following table shows the MIPS instruction format. For more information, see MIPS technical documents [1].

Type	format (bits)						
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)	
I	opcode (6)	rs (5)	rt (5)	immediate (16)			
J	opcode (6)	address (26)					

**Table 1 : MIPS Instruction format**

The Data address space is 4kB. Memory latency will be according to Table 2

Memory	Write Latency	Read Latency
Program Memory (I-Cache)	1 clk	1 clk
Data Memory (D-Cache)	1 clk	1 clk

**Table 2 : Memory sizes and latency**

## 7. Compiler, Simulator, and Memory

The MARS compiler and simulator, or any other can be used to compile and simulate the MIPS assembly code. MARS compiler can also export binary file of the memory code segment and data segment contents into files in different format versions to be loaded to ITCM and DTCM of any softcore implemented on FPGA. It can also simulate a cache performance and more.

The mars compiler, installation instructions and documentation are available at:

<http://courses.missouristate.edu/KenVollmar/MARS/>

## 8. CPU core design verification and QA test

- a) Find the fmax and the critical path of single-cycle and pipelined MIPS architectures, Explain the factors that influence these results.
- b) Functional verification using ModelSim and Memory list property.
- c) Electrical verification using Signal Tap and ISMCE properties (together and separately) in Quartus based on the FPGA board.
- d) As a QA test, you need to write assembly code, which compiles the next C code.  
At the end, calculate **IPC** using **CLKCNT, STCNT, FHCNT** Registers and compare the results to the predicted manual **IPC** theoretical calculation.

**Note:** see on Moodle a file that contains the Required Support of the MIPS instruction set.

```
#define M 4

void addMats(int Mat1[M][M], int Mat2[M][M], int resMat[M][M]){
    define it yourself ...
}

void main(){ //int=32bit
    int Mat1[M][M]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
    int Mat2[M][M]={{13,14,15,16},{9,10,11,12},{5,6,7,8},{1,2,3,4}};
    int resMat[M][M];

    addMats(Mat1,Mat2,resMat); // resMat = Mat1 + Mat2
}
```

- e) Make an advanced QA Test by writing at least two assembly source code to measure the performance of your pipelined MIPS CPU design. Use various instructions from the required instructions set. Simulate, verificate the IPC and make a comparison with the theoretical calculation where all hazard kinds are marked on handwritten source code (the accuracy of IPC matters).

## 9. Requirements

You must do the following tasks:

- ModelSim Simulation with maximal coverage.
- Analyze the critical path, explain where it is in your VHDL design and find the maximal operating clock.
- Load the design onto the FPGA and verify the simulation results.
- **Run the required assembly source codes and explore them.**

The following must be presented in **Lab5.pdf** report file.

1. Top level block review diagram of your design.
2. For each block in the top-level design:
  - RTL Viewer results
  - Logic usage for each block (Combinational and Flip-Flops).
  - Graphical description (a square with ports going in and out).
  - Port Table (direction, size, functionality).
  - Short description.
3. Maximum (Critical) path of your design – explain where it is in the code and how it is possible to optimize if you would have more time. What is the maximum clock frequency?
4. Minimum path analysis.
5. Documentation Style - Content with page numbers, Images and tables will be numbered. The caption of an images and tables below the images or tables.
6. Elaborated analysis and wave forms:
  - Maximal Frequency and critical paths from Timing Analyzer.
  - Proof of work using Signal Tap shot screens.  
**Recall that, proof of work using Signal Tap is mandatory.**
  - One basic waveform to explain the system timing.

### Design requirements:

1. The design must be well commented.
2. The system must work from only one clock.
3. System RESET (KEY0) must be synchronous.
4. Conclusions
5. A ZIP file in the form of **id1\_id2.zip** (where id1 and id2 are the identification number of the submitters, and  $id1 < id2$ ) **must be upload to Moodle only by student with id1** (any of these rules violation disqualifies the task submission).
6. The ZIP file will contain the next six subdirectories (**only the exact next sub folders**):

Directory	Contains	Comments
VHDL	Project VHDL files	Only VHDL files, excluding test bench  <b>Note: your project files must be well compiled (in ModelSim and Quartus separately) without errors as a basic condition before submission</b>
TB	VHDL files that are used for test bench	Only one <b>tb.vhd</b> for the overall DUT
SIM	ModelSim DO files	Only for <b>tb.vhd</b> of the overall DUT
DOC	Project documentation	<b>Readme.txt</b> and <b>Lab5.pdf</b> full report file
Quartus	<ul style="list-style-type: none"> <li>• Signal Tap files used in project verification</li> <li>• Project SOF file</li> <li>• Project SDC file</li> </ul>	<b>Do not place files that are not relevant for compilation or is a result of compilation!</b>
CODE	The assembly source code of clauses 8d,8e	

Table 3 : Directory Structure

## 10. Grading policy

Weight	Task	Description
10%	Documentation	The "clear" way in which you presented the requirements, the analysis, and the conclusions on the work you've done
90%	Analysis and Test	The correct analysis of the system (under the requirements)

Table 4: Grading

**Late submissions will not be received.**

## 11. References

- [1]. MIPS32® Architecture for Programmers Volume I to III (from Moodle under Final Project)
- [2]. ALTPPLL User Guide: [http://www.altera.com/literature/ug/ug\\_altppll.pdf](http://www.altera.com/literature/ug/ug_altppll.pdf)
- [3]. Altera RAM user guide: [http://www.altera.com/literature/ug/ug\\_ram\\_rom.pdf](http://www.altera.com/literature/ug/ug_ram_rom.pdf)
- [4]. Altera MegaFunction User Guide:  
[www.altera.com/literature/ug/ug\\_intro\\_to\\_megafunctions.pdf](http://www.altera.com/literature/ug/ug_intro_to_megafunctions.pdf)
- [5]. Bin2Hex utility
  - 32bit - <http://www.keil.com/download/docs/113.asp>
  - 64bit - <http://www.ht-lab.com/freeutils/bin2hex/bin2hex.html>