CPE 300L
DIGITAL SYSTEM ARCHITECTURE AND DESIGN LABORATORY

LABORATORY 9
MIPS: SINGLE CYCLE IMPLEMENTATION I

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
UNIVERSITY OF NEVADA, LAS VEGAS

OBJECTIVE

Gain experience in designing a complete CPU by implementing and experimenting with a single cycle implementation of a limited subset of MIPS instructions.

INTRODUCTION

MIPS has a Load/Store architecture, meaning the CPU uses registers to perform operations in ALU and only load and store instructions are allowed to access data memory This is typical to RISC (Reduced Instruction Set Architectures) and has certain benefits such as capturing and resolving data dependencies in both Compiler (static) time and the run-time. This is a further topic of your study. Out of two operands that ALU operates on, one can be also an immediate value provided from inside the instruction.
MIPS processors have 32 general purpose registers, but some of these are reserved. A fair number of registers however are available for your use. The CPU state consists of the PC (Program counter), 32 general registers and also Hi and Lo registers that hold the result of multiplication and division. They can be accessed by special instructions (move from Hi and move from Lo). PC contains the address of the next instruction to be executed. As the processor executes the instruction, the program counter is incremented, and fetched from the Instruction memory. The data memory is separate, and in this way, the architecture is so called Harvard architecture (separate memories for instruction and data). Being a representative of RISC, the MIPS doesn't assign individual instructions to complex, logically intensive tasks. This is in contrast to complex instruction set computer (CISC) architectures like x86 which have multimedia operating, cache management, memory allocation and other complex instructions. MIPS and other RISC architectures were based on the philosophy that, among other things, by only implementing a small core (only a few dozen instructions, instead of several hundred) of the most common instructions, architects could simplify the design and speed up the majority of common instructions so much that the cost of implementing complex programs as multiple instructions would be hidden. MIPS implements about 111 instructions, and some of them are pseudo-instructions, hat is available for the programmer, but implemented by means of other instructions. Another feature of MIPS typical to RISC architectures is a small number of instruction formats (three in MIPS) and uniform length of all instructions (4 bytes). Please refer to the text book and the reference material on-line for studying the Instruction set.

The datapath with the control unit of a single cycle implementation of MIPS for a subset of instructions: *add*, *sub*, *addi*, *and*, *or*, *slt*, *beq*, *j* is shown in the following circuit:

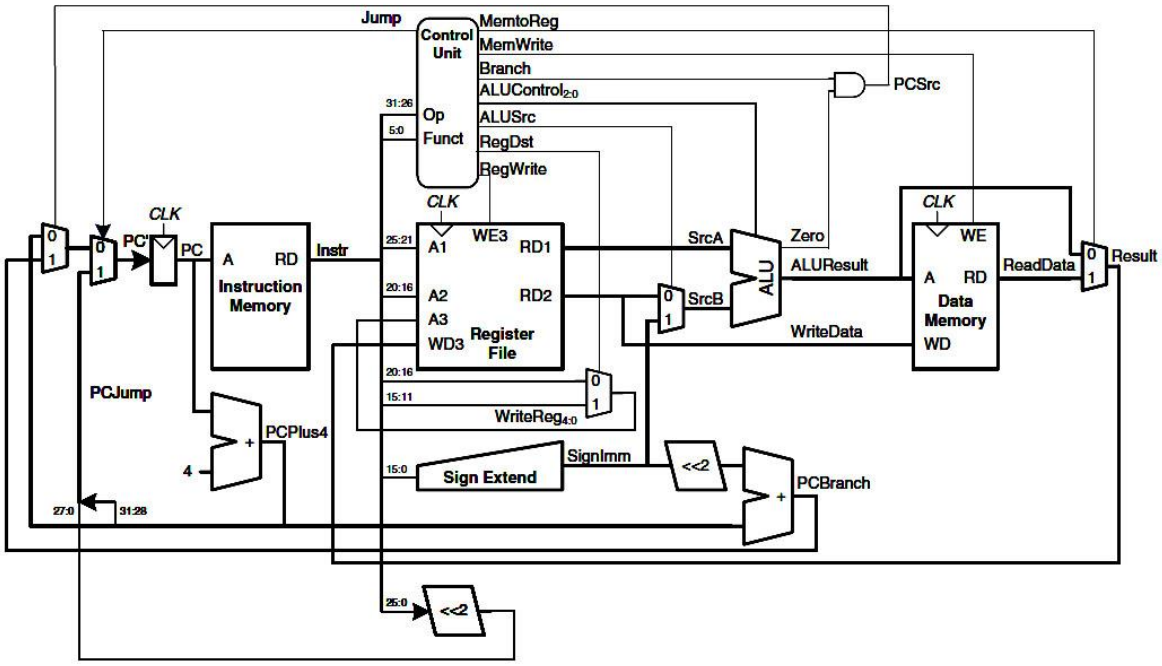

Fig. 1. MIPS single cycle design.

For implementing a subset, we need only few bits rather than 6 bits of the opcode and 6 bits of funct field of *R-type* instructions

The ALU opcode encoding tables in Fig.2 show two-step decoding of the subset. At the first step, we decode ALUop with two bits *00- add*, *01* for *sub*, and *10* for other instructions. Then we look at *funct* field and further decode to obtain three *ALUControl* signals.

ALUOp	Meaning
00	add
01	subtract
10	look at `funct` field
11	n/a

ALU decoder truth table

ALUOp	Funct	ALUControl
00	X	010 (add)
X1	X	110 (subtract)
1X	100000 (add)	010 (add)
1X	100010 (sub)	110 (subtract)
1X	100100 (and)	000 (and)
1X	100101 (or)	001 (or)
1X	101010 (slt)	111 (set less than)

Fig. 2. ALU opcode encoding table

The control unit is decoding the instructions and asserting a set of control signals (blue font in Fig. 3). Although it is shown to be external memory for both instructions and data, we implement it on board for synchronous operation of all parts of the design; otherwise interaction with outside (the board) main memory would asynchronous, that is based on the handshake.

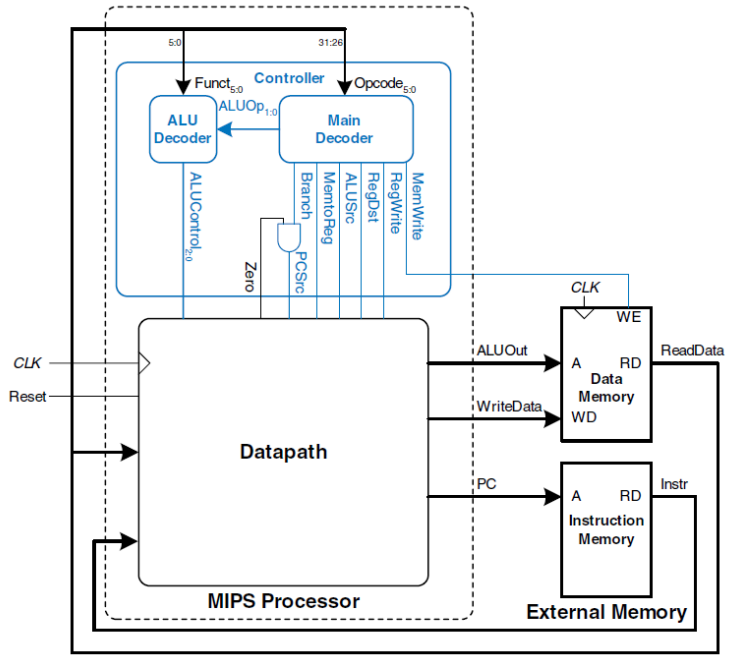

Fig. 3. A single cycle MIPS CPU (Limited set).

Fig.4 shows the control signals (except *ALUControl*)

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp	Jump
R-type	000000	1	1	0	0	0	0	10	0
lw	100011	1	0	1	0	0	1	00	0
sw	101011	0	X	1	0	1	X	00	0
beq	000100	0	X	0	1	0	X	01	0
addi	001000	1	0	1	0	0	0	00	0
j	000010	0	X	X	X	0	X	XX	1

Fig. 4. Control signals per instruction

The design modules can be found at:
https://faculty.unlv.edu/eelabs/docs/labs/cpe300L/06_design_files.zip

LAB DELIVERIES

PRELAB DELIVERABLES:

1. Describe, what is the function of each block (there are 16 blocks) in Fig.1?
2. Study the given code and write the name of module that generates each block (there are 16 blocks) in Fig.1.

LAB EXPERIMENTS

Demonstrate each experiment to TA.

1. Testbench for MIPS subset.

Download all the *v*. files provided:
https://faculty.unlv.edu/eelabs/docs/labs/cpe300L/06_design_files.zip

Implement the test bench on the sample code provided below.

```
# Test the MIPS processor.
# add, sub, and, or, slt, addi, lw, sw, beq, j
# If successful, it should write the value 7 to address 84

#           Assembly                Description                 Address     Machine
main:       addi $2, $0, 5          # initialize $2 = 5           0           20020005
            addi $3, $0, 12         # initialize $3 = 12          4           2003000c
            addi $7, $3, -9         # initialize $7 = 3           8           2067fff7
            or   $4, $7, $2         # $4 <= 3 or 5 = 7            c           00e22025
            and  $5, $3, $4         # $5 <= 12 and 7 = 4          10          00642824
            add  $5, $5, $4         # $5 = 4 + 7 = 11             14          00a42820
            beq  $5, $7, end        # shouldn't be taken          18          10a7000a
            slt  $4, $3, $4         # $4 = 12 < 7 = 0             1c          0064202a
            beq  $4, $0, around     # should be taken             20          10800001
            addi $5, $0, 0          # shouldn't happen            24          20050000
around:     slt  $4, $7, $2         # $4 = 3 < 5 = 1              28          00e2202a
            add  $7, $4, $5         # $7 = 1 + 11 = 12            2c          00853820
            sub  $7, $7, $2         # $7 = 12 - 5 = 7             30          00e23822
            sw   $7, 68($3)         # [80] = 7                    34          ac670044
            lw   $2, 80($0)         # $2 = [80] = 7               38          8c020050
            j    end                # should be taken             3c          08000011
            addi $2, $0, 1          # shouldn't happen            40          20020001
end:        sw   $2, 84($0)         # write adr 84 = 7            44          ac020054
```

It is a machine code, written into the instruction memory (see the *memfile.dat* file). Please note that the instruction memory is limited in this implementation to 255 words. You can see that the address is now defined by 6 bits because two *LSB* 00 for the memory address calculation are appended in the datapath.

Verify the operation of the code in Modelsim.

2. MIPS implementation on DE2

Implement the project on the DE2 board.

- Output the contents of **$2** (*LSByte*) and the memory location of the last instruction (labeled end) in led
- Output also the PC (*LSByte* only) in 7 segment display
- Report the values.

3. TimeQuest analysis

Run the TimeQuest analyzer for checking the cycle time for the designed MIPS (subset).

- Get the report on the cycle time.
- Get the report on the performance of MIPS for the given test program: provide the total time to complete the code: Cycle time x number of instructions.

POSTLAB REPORT

Include the following elements in your postlab report:

<table>
<tr><th>Section</th><th>Element</th></tr>
<tr><td>1</td><td>Theory of operation
Include a brief description of every element and phenomenon that appears during the experiments.
a. What are main elements of single cycle datapath?
b. What kind of circuit is the control unit circuit which generates the control signals; combinational or FSM? Why?
c. Analyze the Instruction set and report what are other functional units that are to be added to ALU. Note, that floating point operations are performed on a Co-processor</td></tr>
<tr><td>2</td><td>Prelab report</td></tr>
<tr><td rowspan="4">3</td><td colspan="2">Results of the experiments</td></tr>
<tr><td>Experiment</td><td>Experiment Results</td></tr>
<tr><td>1</td><td>Testbench code</td></tr>
<tr><td>2</td><td>a. Photos of the DE2 working
b. Output of the test results
c. Binary files of compiled project</td></tr>
<tr><td></td><td>a. Report on the cycle time
b. Report on the performance</td></tr>
<tr><td rowspan="3">4</td><td colspan="2">Answer the questions</td></tr>
<tr><td>Question no.</td><td>Question</td></tr>
<tr><td>1</td><td>What distinguishes MIPS from other processor architectures?</td></tr>
<tr><td></td><td>2</td><td>List 3 advantages of MIPS and 3 disadvantages.</td></tr>
<tr><td>5</td><td colspan="2">Conclusions
Write down your conclusions, things learned, problems encountered during the lab and how they were solved, etc.</td></tr>
</table>

REFERENCES

1. Memory Initialization tutorial:
https://faculty.unlv.edu/eelabs/docs/guides/Memory_initialization_tutorial.pdf

2. Laboratory design files
https://faculty.unlv.edu/eelabs/docs/labs/cpe300L/06_design_files.zip

3. *Using TimeQuest Timing Analyzer*, Altera.
https://faculty.unlv.edu/eelabs/docs/guides/Altera_Timequest.pdf

4. MIPS Quick Reference,
https://faculty.unlv.edu/eelabs/docs/guides/MIPS_Quick_Reference.pdf

5. MIPS Opcodes, https://faculty.unlv.edu/eelabs/docs/guides/MIPS_Opcodes.pdf

6. MIPS Assembly/MIPS Architecture:
https://en.wikibooks.org/wiki/MIPS_Assembly/MIPS_Architecture