

## CMPEN 331 – Computer Organization and Design, Lab 3

**Due Sunday October 27, 2019 at 11:59 pm (Drop box on Canvas)**

This lab introduces the idea of the pipelining technique for building a fast CPU. The students will obtain experience with the design implementation and testing of the first two stages (Instruction Fetch, Instruction Decode) of the five-stage pipelined CPU using the Xilinx design package for FPGAs. It is assumed that students are familiar with the operation of the Xilinx design package for Field Programmable Gate Arrays (FPGAs) through the Xilinx tutorial available in the class website.

### 1. Pipelining

Pipelining is an implementation technique in which multiple instructions are overlapped in execution. The five-stage pipelined CPU allows overlapping execution of multiple instructions. **Although an instruction takes five clock cycle to pass through the pipeline, a new instruction can enter the pipeline during every clock cycle.** Under ideal circumstances, the pipelined CPU can produce a result in every clock cycle. Because in a pipelined CPU there are multiple operations in each clock cycle, we must save the temporary results in each pipeline stage into pipeline registers for use in the follow-up stages. We have five stages: IF, ID, EXE, MEM, and WB. The PC can be considered as the first pipeline register at the beginning of the first stage. We name the other pipeline registers as IF/ID, ID/EXE, EXE/MEM, and MEM/WB in sequence. In order to understand in depth how the pipelined CPU works, we will show the circuits that are required in each pipeline stage of a baseline CPU.

### 2. Circuits of the Instruction Fetch Stage

The circuit in the IF stage are shown in Figure 2. Also, looking at the first clock cycle in Figure 1(b), the first lw instruction is being fetched. In the IF stage, there is an instruction memory module and an adder between two pipeline registers. The left most pipeline register is the PC; it holds 100. In the end of the first cycle (at the rising edge of clk), the instruction fetched from instruction memory is written into the IF/ID register. Meanwhile, the output of the adder ( $PC + 4$ , the next PC) is written into PC.

### 3. Circuits of the Instruction Decode Stage

Referring to Figure 3, in the second cycle, the first instruction entered the ID stage. There are two jobs in the second cycle: to decode the first instruction in the ID stage, and to fetch the second instruction in the IF stage. The two instructions are shown on the top of the figures: the first instruction is in the ID stage, and the second instruction is in the IF stage. The first instruction in the ID stage comes from the IF/ID register. Two operands are read from the register file (Regfile in the figure) based on rs and rt, although the lw instruction does not use the operand in the register rt. The immediate (imm) is sign-extended into 32 bits. The regrt signal is used in the ID stage that selects the destination register number; all others must be written into the ID/EXE register for later use. At the end of the second cycle, all the data and control signals, except for regrt, in the ID stage are written into the ID/EXE register. At the same time, the PC and the IF/ID register are also updated.

2nd:  
IF/ID  
ID/EXE  
EXE  
MEM  
WB

1st pipeline  
Registers:  
IF  
ID  
EXE  
MEM  
WB

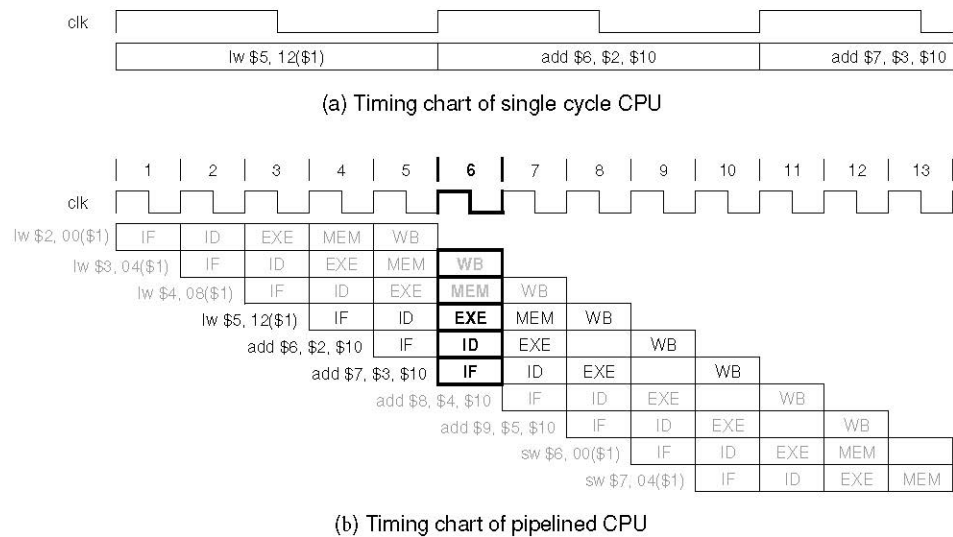


Figure 1 Timing chart comparison between two types of CPUs

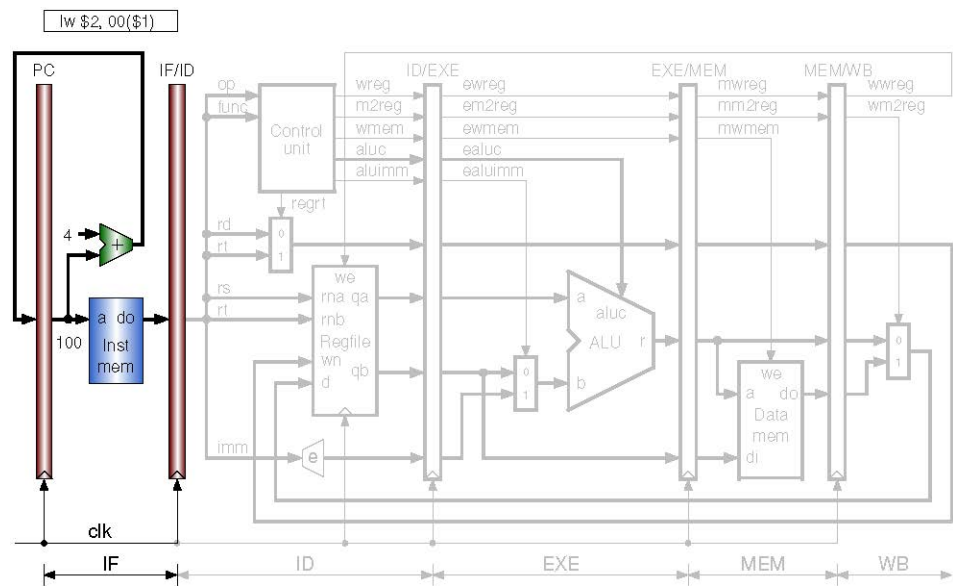


Figure 2 Pipeline instruction fetch (IF) stage

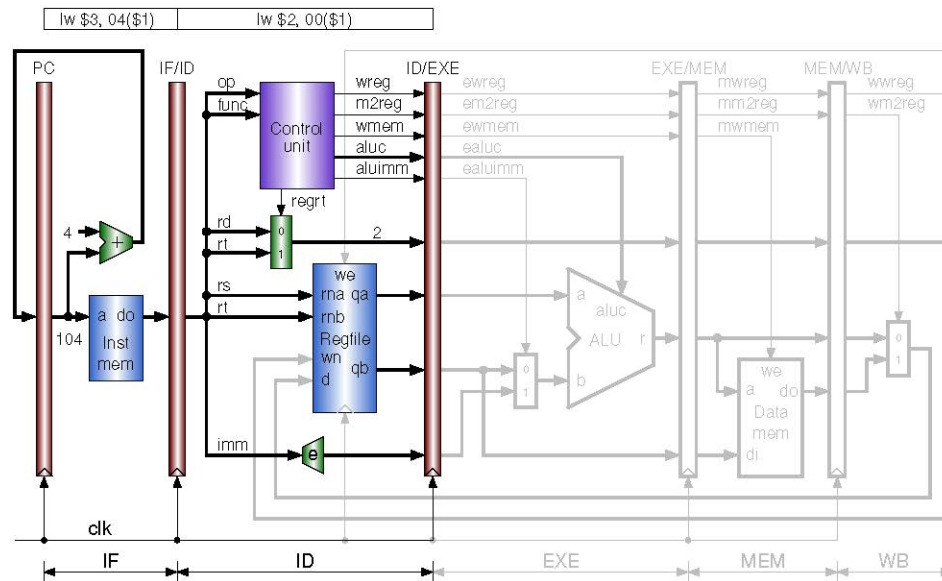


Figure 3 Pipeline instruction decode (ID) stage

4. Table 1 lists the names and usages of the 32 registers in the register file.

Table 1 MIPS general purpose register

Register Name	Register Number	Usage
\$zero	0	Constant0
\$at	1	Reserved for assembler
\$v0, \$v1	2, 3	Function return values
\$a0 - \$a3	4 - 7	Function argument values
\$t0 - \$t7	8 - 15	Temporary (caller saved)
\$s0 - \$s7	16 - 23	Temporary (callee saved)
\$t8, \$t9	24, 25	Temporary (caller saved)
\$k0, \$k1	26, 27	Reserved for OS Kernel
\$gp	28	Pointer to Global Area
\$sp	29	Stack Pointer
\$fp	30	Frame Pointer
\$ra	31	Return Address

5. Table 2 lists some MIPS instructions that will be implemented in our CPU

Table 2 MIPS integration instruction

Inst.	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	Meaning
add	000000	rs	rt	rd	00000	100000	Register add
sub	000000	rs	rt	rd	00000	100010	Register subtract
and	000000	rs	rt	rd	00000	100100	Register AND
or	000000	rs	rt	rd	00000	100101	Register OR
xor	000000	rs	rt	rd	00000	100110	Register XOR
sll	000000	00000	rt	rd	sa	000000	Shift left
srl	000000	00000	rt	rd	sa	000010	Logical shift right
sra	000000	00000	rt	rd	sa	000011	Arithmetic shift right
jr	000000	rs	00000	00000	00000	001000	Register jump
addi	001000	rs	rt		Immediate		Immediate add
andi	001100	rs	rt		Immediate		Immediate AND
ori	001101	rs	rt		Immediate		Immediate OR
xori	001110	rs	rt		Immediate		Immediate XOR
lw	100011	rs	rt		offset		Load memory word
sw	101011	rs	rt		offset		Store memory word
beq	000100	rs	rt		offset		Branch on equal
bne	000101	rs	rt		offset		Branch on not equal
lui	001111	00000	rt		immediate		Load upper immediate
j	000010			address			Jump
jal	000011			address			Call

6. Initialize the first 10 words of the **Data** memory with the following HEX values:

A00000AA  
10000011  
20000022  
30000033  
40000044  
50000055  
60000066  
70000077  
80000088  
90000099

7. Write a Verilog code that implement the following instructions using the design shown in Figure 2 and Figure 3. Write a Verilog test bench to verify your code: (You have to show all the signals written into the IF/ID register and the ID/EXE register in your simulation outputs)

```
# address  instruction          comment
100:      lw $v0, 00($at)      # $2 ← memory[$1+00]; load x[0]
104:      lw $v1, 04($at)      # $3 ← memory[$1+04]; load x[1]
```

Assume that the register \$at has the value of 0

8. Write a report that contains the following:
- Your Verilog design code. Use:
    - Device: XC7Z010- CLG400 -1
  - Your Verilog® Test Bench design code. Add “timescale 1ns/1ps” as the first line of your test bench file.

- c. The waveforms resulting from the verification of your design with simulation showing all the signals written into the IF/ID register and the ID/EXE register.
  - d. The design schematics from the Xilinx synthesis of your design. Do not use any area constraints.
  - e. Snapshot of the I/O Planning and
  - f. Snapshot of the floor planning
9. REPORT FORMAT: Free form, but it must be:
- g. One report per student.
  - h. Have a cover sheet with identification: Title, Class, Your Name, etc.
  - i. Using Microsoft word and it should be uploaded in word format not PDF. If you know LaTeX, you should upload the Tex file in addition to the PDF file.
  - j. Double spaced

**10. You have to upload the whole project design file zipped with the word file.**