German University in Cairo

CSEN 601 Computer System Architecture

Project Report

Submitted By:

Ahmed Saleh	28-13165	T-12
Mohamed Mostafa	28-9771	T-15
Abdelrahman El-Kady	28-9799	T-15
Ali Hassan	28-11797	T-16
Mohamed Hussein	28-643	T-15
Abdelrahman Mahmoud	28-10500	T-15

Submitted To: Jailan Saleh

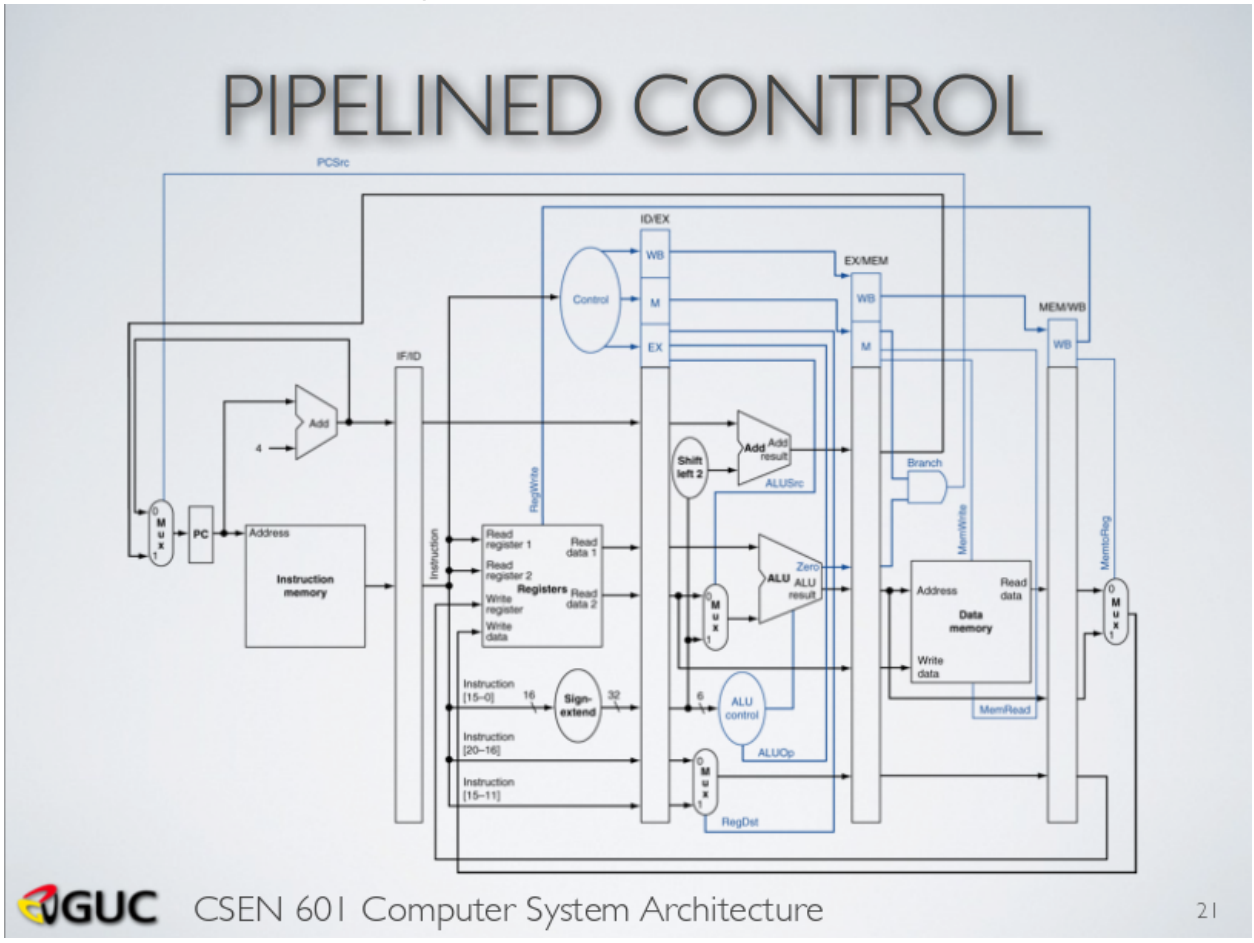
Team Number: 14

Our approach to simulate the Mips Assembly language was to use the concept of multi cycle. In this approach we showed the 5 stages that each instruction go through:
-IF: Instruction fetch and in this stage we read the input from the user and convert the input to binary code to make it easier for the following stages to function.
-ID: Instruction decode and this stage we read the binary code and define the instruction whether it is an R-Format, I-Format or J-Format.
-EX: Perform Alu operations on register values or calculate branched address in case of a jump instruction then sends the data to the MEM Register.
-MEM: Load data from a specific memory address or store data in this location/address.
-WB : In this stage the result will be passed to the destination register in case of r-format instructions or to the target register in case of i-format instruction.

A Graphical user interface is implemented that supports :

- text editor
- compile and run the code
- register values after execution

We used the same data path in the lecture 9 slide 3 :



An additional control signals were used such as :

memByte Determines how the destination register is specified
unsigned Determines whether the loading
compOne Determines whether to compare with one or zero
jump,branch Determine what to be loaded to PC next instruction :
1)incremented PC (normal) 0x00
2) incremented PC + offset (branch) 0x01
3) jump address , 26 to 32 jump address 0x10
4) 32 register address 0x11

---

ins aluop funct aluAction aluControlInput
---
lw 00 xx xxxx add 010
sw 00 xx xxxx add 010 //they all have
lb 00 xx xxxx add 010 // the same
sb 00 xx xxxx add 010 // Alu inputs .
lbu 00 xx xxxx add 010 // however they are diffrent in control signals
lui
---
beq 01 xx xxxx sub 110
bne 01 xx xxxx sub 110 //diffrence in control signals .. check the compOne signal
---
add 10 10 0000 add 010
sub 10 10 0010 sub 110
addi 10 10 0000 add 010
sll 10 01 0000 sll 101 //i added that because it wasn't supported
srl 10 00 0010 srl 011 //i added that as well
and 10 10 0100 and 000
nor 10 10 0111 nor 100 //and that too
slt 10 10 1010 slt 111
sltu 10 10 1011 sltu 001 // and also this one

****

control signals
ALUSrc Selects the second source operand for the ALU (rt or sign-extended immediate field in Patterson and Hennessey).
ALUOp Either specifies the ALU operation to be performed or specifies that the operation should be determined from the function bits.
MemRead Enables a memory read for load instructions.
MemWrite Enables a memory write for store instructions.
RegWrite Enables a write to one of the registers.
RegDst Determines how the destination register is specified (rt or rd in Patterson and Hennessey).

MemtoReg Determines where the value to be written comes from (ALU result or memory in Patterson and Hennessey).
memByte Determines how the destination register is specified (rt or rd in Patterson and Hennessey).
unsigned Determines whether the loading
compOne determines whether to compare with one or zero
jump,branch determine what to be loaded to PC next instruction
incremented PC (normal) 0x00 , incremented PC + offset (branch) 0x01, jump address , 0x10 26 to 32 jump address ,0x11 32 register address

We distributed the work among us as follows:
Mohamed Mostafa worked on the fetching stage, GUI and the parser.
Ali Hassan worked on decoding the instructions stage.
Abdelrahman El-Kady worked on building the structure for the whole project and modifying the new control signals.
Ahmed Saleh worked on the WriteBack stage.
Mohamed Hussein worked on the execute stage and writing the report.
Abdelrahman Mahmoud worked on the loading/storing in the memory stage and writing the report.