THE UNIVERSITY OF KANSAS

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

EECS 645 – Computer Architecture

Fall 2019

Final Project (Single-Cycle MIPS)

Student Name: Student ID:

Final Project

In this project you will be designing the Single-Cycle (non-pipelined) version of the MIPS processor that supports the following subset, i.e., **16 instructions**, of MIPS ISA:

a) 9 Arithmetic/Logical instructions: *add, sub, and, or, nor, slt, addi, ori, lui*
b) 2 Memory reference: *lw, sw*
c) 5 Control transfer: *beq, bne, j, jal, jr*

The processor has the following interface:

- Inputs
  - Clock (clk → 1 bit)
  - Asynchronous reset for processor initialization and for mimicking program load (rst → 1 bit)

You are required to:

a) Modify and neatly regenerate/redraw the provided microarchitecture (datapath and control path) shown in Figure 1 such that your design supports the above 16 instructions.
   - *Submit the modified diagram with your Vivado project.*

b) Extend the ALU functionality as shown in Figure 2.

c) Complete the provided truth table, shown in Figure 3, of the MIPS control unit that supports the above 16 instructions.
   - *Submit the completed table with your Vivado project.*

d) Describe in structural and behavioral VHDL the modified microarchitecture (datapath and control path) of this version of the MIPS processor.

e) Verify the correct operation through Vivado Simulator by comparing your simulation results with those of MARS runs using the provided test programs.

Your memory address translation/mapping should follow the convention shown in Figure 4.

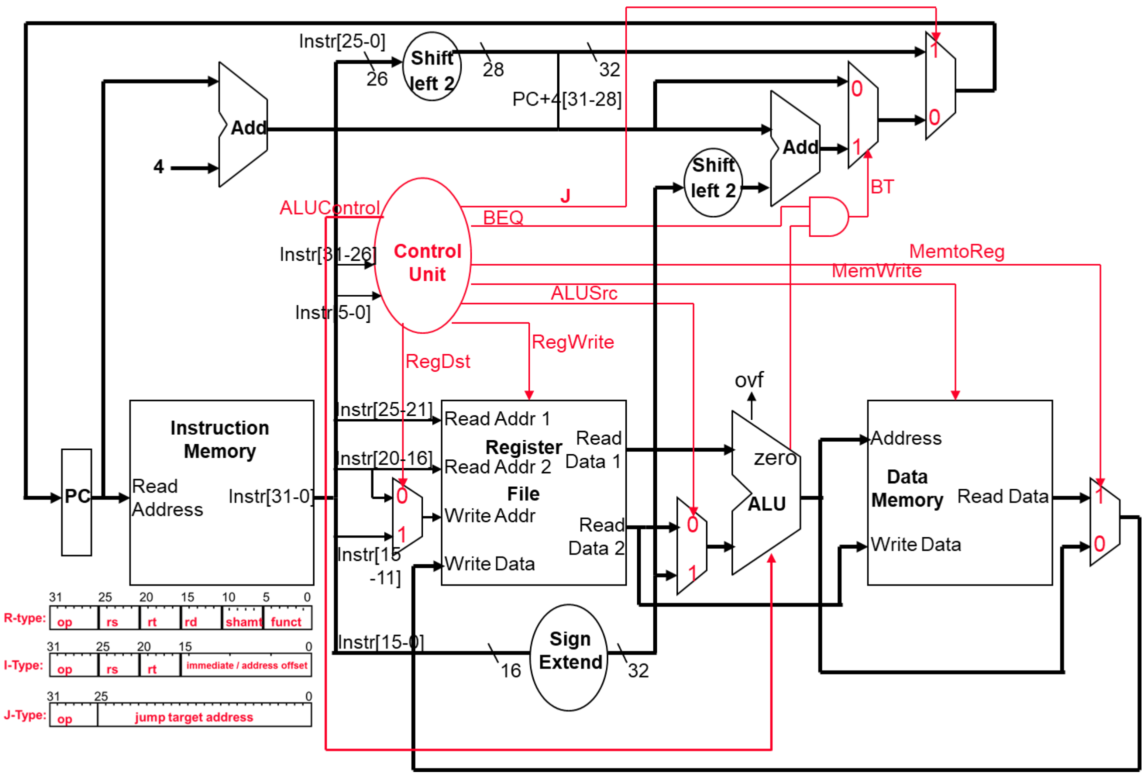


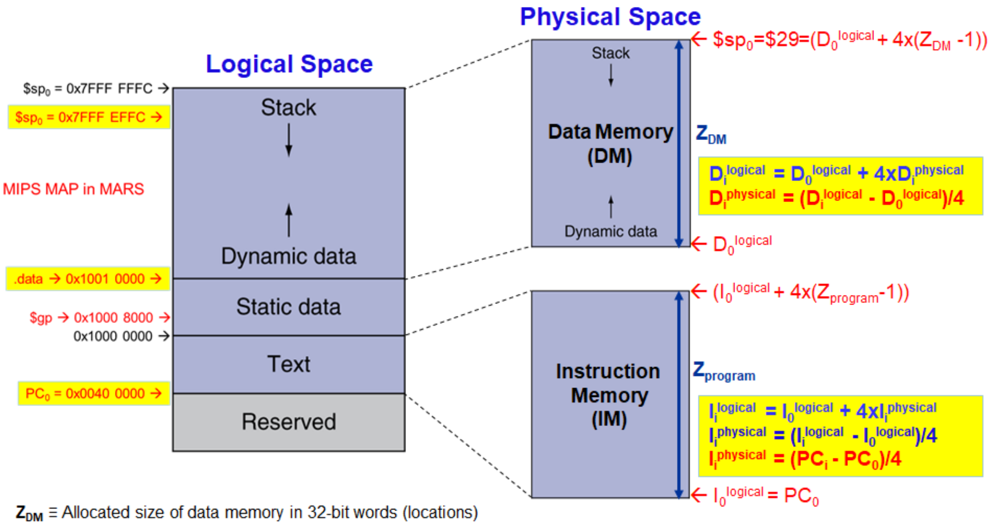
Figure 1. Single-Cycle (non-pipelined) MIPS microarchitecture supporting 11 instructions

Operation	opcode	funct	ALU function	ALUControl
AND	R-type ≡ 000000	100100	AND	0000
OR		100101	OR	0001
add		100000	add	0010
subtract		100010	subtract	0110
set-on-less-than		101010	set-on-less-than	0111
NOR		100111	NOR	1100
add immediate	addi ≡ 001000	xxxxxx	add	0010
load word	lw ≡ 100011			
store word	sw ≡ 101011			
branch on equal	beq ≡ 000100	xxxxxx	subtract	0110
branch not equal	bne ≡ 000101			
or immediate	ori ≡ 001101	xxxxxx	OR immediate	1101
load upper immediate	lui ≡ 001111	xxxxxx	shift left 16	1111

Figure 2. Extended ALU functionality supporting 16 instructions

opcode	**funct**	**ALUControl**	**RegDst**	**ALUSrc**	**MemToReg**	**RegWr**	**MemWr**	**BEQ**	**J**	**BNE**	**Jal**	**Jr**
R-type ≡ 000000	**AND** 100100	**0000**	**1**	**0**	**0**	**1**	**0**	**0**	**0**			
	OR 100101	**0001**										
	add 100000	**0010**										
	sub 100010	**0110**										
	slt 101010	**0111**										
	NOR 100111	**1100**										
	jr 001000											
lw ≡ 100011	xxxxxx	**0010**	**0**	**1**	**1**	**1**	**0**	**0**	**0**			
sw ≡ 101011	xxxxxx	**0010**	**0**	**1**	**0**	**0**	**1**	**0**	**0**			
beq ≡ 000100	xxxxxx	**0110**	**0**	**0**	**0**	**0**	**0**	**1**	**0**			
j ≡ 000010	xxxxxx	**0000**	**0**	**0**	**0**	**0**	**0**	**0**	**1**			
addi ≡ 001000	xxxxxx	**0010**	**0**	**1**	**0**	**1**	**0**	**0**	**0**			
bne ≡ 000101	xxxxxx											
jal ≡ 000011	xxxxxx											
ori ≡ 001101	xxxxxx											
lui ≡ 001111	xxxxxx											

Figure 3. MIPS control unit supporting 16 instructions


Logical Space
Physical Space
$\$sp_0$ = 0x7FFF FFFC →
$\$sp_0$ = 0x7FFF EFFC →
MIPS MAP in MARS
.data → 0x1001 0000 →
$gp → 0x1000 8000 →
0x1000 0000 →
PC_0 = 0x0040 0000 →
Stack
Dynamic data
Static data
Text
Reserved
Stack
Data Memory (DM)
Dynamic data
Instruction Memory (IM)
← $\$sp_0 = \$29 = (D_0^{logical} + 4x(Z_{DM} - 1))$
Z_{DM}
$D_i^{logical} = D_0^{logical} + 4xD_i^{physical}$
$D_i^{physical} = (D_i^{logical} - D_0^{logical})/4$
← $D_0^{logical}$
← $(I_0^{logical} + 4x(Z_{program} - 1))$
$Z_{program}$
$I_i^{logical} = I_0^{logical} + 4xI_i^{physical}$
$I_i^{physical} = (I_i^{logical} - I_0^{logical})/4$
$I_i^{physical} = (PC_i - PC_0)/4$
← $I_0^{logical} = PC_0$

Z_{DM} ≡ Allocated size of data memory in 32-bit words (locations)
$D_0^{logical}$ ≡ First address of data memory, could be anything, e.g., 0 or **0x1001 0000**

$Z_{program}$ ≡ Allocated size of instruction memory = Size of test program in 32-bit words (instructions)
$I_0^{logical}$ ≡ First address of instruction memory, could be anything, e.g., 0 or **0x0040 0000 = PC_0**

Figure 4. Convention and formulae for Memory Address Translation/Mapping

- Steps:
  1) Download the file “Project_MIPS_Single_Cycle.zip” from blackboard and extract its contents.
  2) Rename the folder “Project_MIPS_Single_Cycle” to “Project_MIPS_Single_Cycle_<your last name>”, for example “Project_MIPS_Single_Cycle_El-Araby”.
  3) Launch Vivado and create a new project, for example “vivado_project”, with the default settings under the following directory “\Project_MIPS_Single_Cycle_<your last name>” resulting in the following project directory “\Project_MIPS_Single_Cycle_<your last name>\vivado_project\”
  4) Add to the project the VHDL design and simulation source files from the folders; “\Project_MIPS_Single_Cycle_<your last name>\design_sources” and “\Project_MIPS_Single_Cycle_<your last name>\simulation_sources” respectively.
  5) Edit the VHDL files in the folder “\Project_MIPS_Single_Cycle_<your last name>\design_sources\” according to your design such that it describes the required *MIPS microarchitecture*.
  6) Set the simulation time to the proper time, e.g., ***5,200 μs***, and then launch Vivado Simulator.
  7) Verify the correctness of your design by comparing your Vivado simulation results with those of MARS runs using one or both of the provided assembly test programs “\Project_MIPS_Single_Cycle\test_program\fibonacci_sequence_non-recursive.asm” and “\Project_MIPS_Single_Cycle\test_program\fibonacci_sequence_recursive.asm”. You may go back to step 5 to correct your code until your design works properly as required.
  8) Attach the modified microarchitecture diagram and the completed control unit truth table to the folder “\Project_MIPS_Single_Cycle_<your last name>”.
  9) After you are done, compress the folder “\Project_MIPS_Single_Cycle_<your last name>” to “Project_MIPS_Single_Cycle_<your last name>.zip”, for example “Project_MIPS_Single_Cycle_El-Araby.zip” and upload it to blackboard before the due date and time.

Important Hint:

When you use the components “PC_register”, “InstrMem”, “RegFile”, “ALU”, “DataMem”, and “CU”, in your top-level design “top_mips_single_cycle.vhd”, name their instances by appending “_inst” to the end of the component name as follows “PC_register_inst”, “InstrMem_inst”, “RegFile_inst”, “ALU_inst”, “DataMem_inst”, and “CU_inst”. This will guarantee the correct setup of your waveform configuration for showing all needed signals. For example:

```
PC_register_inst : PC_register
 PORT MAP ( PC_next    => . . . ,
            clk        => . . . ,
            rst        => . . . ,
            PC_current =>  . . .
 );
```

Grade Distribution:

- Functional Correctness, i.e., correct source code → 75 / 100
- Proper Setup of Vivado Project → 25 / 100

NOTE:

Homework submission is a "Single Attempt", i.e., carefully review everything that you want to submit before hitting the "submit" button and make sure that you have uploaded all documents you want to submit and have not missed anything.