NANYANG TECHNOLOGICAL UNIVERSITY

SC3050 Advanced Computer Architecture

LAB-3

There are two phases in lab 3. In phase 1, you will use the datapath designed in lab 2 along with the given Verilog code of a program counter and instruction memory to create a simple four-stage pipelined CPU for execution of register (R) type instructions. The RF has thirty-two registers each having a bit-width of 64 (as in Lab 2). In this lab you will simulate the four-stage pipelined implementation of R-type instructions and understand its functionality. You will be provided with testbench code of four stage pipelined implementation for R type instructions. You are also asked to find the area and time complexity of the four-stage pipelined architecture. In phase 2 of this lab, you will implement a five-stage pipelined CPU architecture including data memory to incorporate D-type instructions (STUR and LDUR). You will also simulate and analyze the functionality of the five-stage pipeline along with its area and time complexity.

I. FOUR-STAGE PIPELINED IMPLEMENTATION OF CPU (FOR REGISTER TYPE INSTRUCTIONS ONLY)

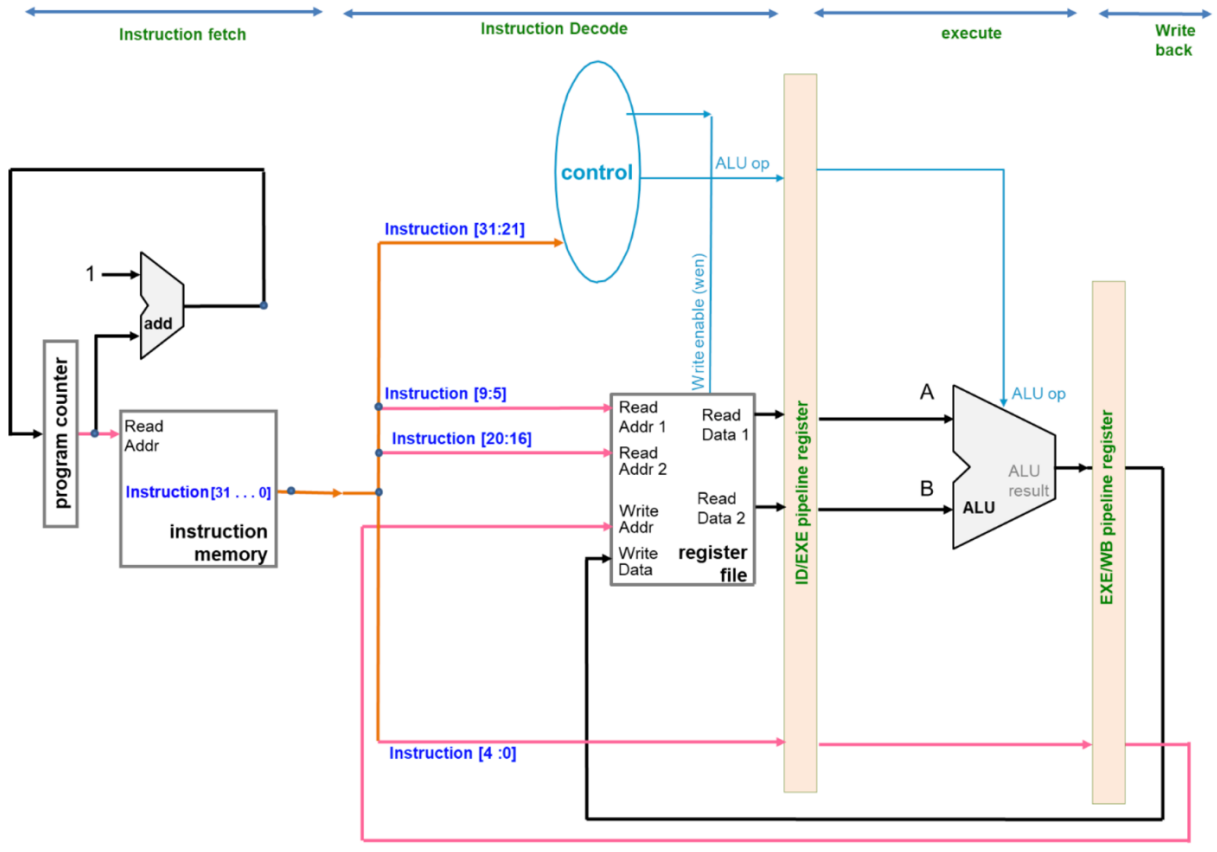

Fig.1 Four-stage pipelined CPU (R-type instructions).

In order to understand the advantages of pipelining a simple four-stage pipeline is explained here. The basic block diagram is given in Fig.1.

The Program counter (PC) is given in "PC.v" points to the next instruction using an incrementor.

The address of the instruction to be executed is fed into instruction memory "imemory.v" from PC. The width of the address of your processor is 64-bits; each memory location is of 32-bits width (in this case PC will increment by 1 to point to the next instruction). The instruction memory has a 64-bit address port and gives 32-bit instruction in the referenced location as output without any control signals. The read operation is possible anytime for instruction memory; the access time is one clock cycle. The Verilog module for the instruction memory "imemory.v" is given to you. The instructions are preloaded to instruction memory with the help of a text file by the name "imem_testw.mem". Each instruction is written as per the 32-bit instruction format explained in lab 2 for the R type instruction. Note that memory is clocked and hence we have a clock cycle delay for getting the output from memory.

The four stages are fetch, decode, execute and write back. In the first stage, program counter provides address to instruction memory. In the second stage, the instruction is read from instruction memory and instruction is decoded to generate the control signals. The operand addresses are connected directly to the address ports of RF for decoding the desired register values. The third stage comprises of execution and fourth stage is write back. The values from register file goes to ALU in the execute stage and the ALU operation is specified by the opcode of the instruction (generated by the control logic and provided to the ALU). Finally, the ALU output is written back to the register file in the write back stage.

The Verilog file "pipelined_regfile_4stage.v" (provided) is the top module of the four-stage pipelined CPU. This top module consists (instantiates) of "PC.v", "regfile.v", "control.v", "imemory.v", "alu.v", "ID_EXEstage.v" and "EXE_WBstage.v". The "ID_EXEstage.v" is the pipeline register between decode and execute section and "EXE_WBstage.v" is the pipeline register between execute and write back section.

Please note that as there are no load and store instructions implemented in this datapath, we are unable to initialize the registers in the register file from the memory. Hence we initialize the registers (hardcoding) inside "register.v"(please note regdata[1] is initialized as 3 and regdata[2] is initialized as 2 in "regfile.v").

1. Please note that the "write enable (wen)" of the register file is not taken to the pipeline (ID/EXE) in the given code because for all your instructions (R type) "write enable (wen) =1". Hence the circuit is always write-enabled, and "wen" is no longer a variable. But if you have instruction like store and branch where you don't want data to be written to a "regfile" you may have to take "write enable" signal as a variable and need to take it through pipeline.

2. Double click RTL schematic to view the circuit diagram for the four-stage pipelined implementation. The RTL schematic should closely resemble Fig.1.

3. Test the four-stage pipelined CPU implementation using the test bench given "test_bench_4_stage_pipeline.v". Note the operation and analyze the same. The "imemory.v" uses file operation to get the instruction and it reads "imem_testw.mem" as shown in Fig.2 as per the address given by PC. Before running the test, make sure to copy "imem_testw.mem" into the directory: "...\working folder\sim_1\behav\xsim". Verify the operation. Did you find any anomaly in the result? Why is it? What would be the method to resolve the same? Can No operation instruction (NOP) help? You can add more inputs to fully check the functionality.

4. Note the cycle changes in the testbench for each instruction as well as the changes for the destination register content once you execute one instruction.

64 bit PC address (in hex)	32 bit instruction from IMEM	meaning
0000000000000000	001F03FF	ADD X31, X31, X31 (NOP)
0000000000000001	00040023	ADD X3, X1, X4
0000000000000002	00220024	SUB X4, X1, X2
0000000000000003	00440027	AND X7, X1, X4
0000000000000004	006200E8	XOR X8, X7, X2
0000000000000005	001F03FF	ADD X31, X31, X31 (NOP)
0000000000000006	001F03FF	ADD X31, X31, X31 (NOP)

Fig.2: R type instructions in the IMEM text file

5. To verify the operation and see the content of the register file after simulation, you can use the Tcl Console to get access to the register data. Note that the values of register data are in the "regdata" array, you may first find where this array is located by using the following command in Tcl Console:

```
get_objects /test_bench_4_stage_pipeline/uut/RF0/*
```

You may see the "regdata" is listed in the displayed paths as "/test_bench_4_stage_pipeline/uut/RF0/regdata", then you can obtain the values inside "regdata" by running the following commands **one-by-one**:

```
set NREG 32

set scope_path "/test_bench_4_stage_pipeline/uut/RF0"

for {set i 0} {$i < $NREG} {incr i} {
    set reg_value [get_value ${scope_path}/regdata[$i]]
    puts "regdata[$i] = $reg_value"
}
```

You can see the contents of the "regdata" printed as the following:

```
regdata[0] = 0000000000000000
regdata[1] = 0000000000000003
regdata[2] = 0000000000000002
regdata[3] = 0000000000000003
regdata[4] = 0000000000000001
regdata[5] = 0000000000000000
regdata[6] = 0000000000000000
regdata[7] = 0000000000000001
regdata[8] = 0000000000000003
regdata[9] = 0000000000000000
regdata[10] = 0000000000000000
regdata[11] = 0000000000000000
regdata[12] = 0000000000000000
regdata[13] = 0000000000000000
regdata[14] = 0000000000000000
regdata[15] = 0000000000000000
regdata[16] = 0000000000000000
```

6. You can use https://personal.ntu.edu.sg/smitha/OPCoder/OPCoder/converter.html to convert the ARM instructions to machine code in binary or hexadecimal number system.

EVALUATION - 1

1) Synthesize the four-stage pipelined CPU for R type instructions. Find the LUT consumption, number of registers and minimum clock period.

CPU	No of LUT slices	No of slice registers	Minimum Clock Period
Four stage pipeline			

Table 1: Slices and delay for four-stage pipelined CPU implementation.

(Please make sure to implement all the experiments and fill out Table 1 in the last page of this manual. You'll need to submit the outcomes at the end of this lab.)

II. FIVE STAGE PIPELINED IMPLEMENTATION OF R & D TYPE CPU

Here we modify the earlier pipeline to include LDUR, and STUR instructions (D-type instructions) along with R-type instructions and convert that to a 5-stage pipelined processor. You will be provided with Verilog code for the datapath. You need to understand the functionality of the 5-stage CPU using the test bench. The datapath for the five-stage pipeline including STUR and LDUR is shown in Fig. 3.

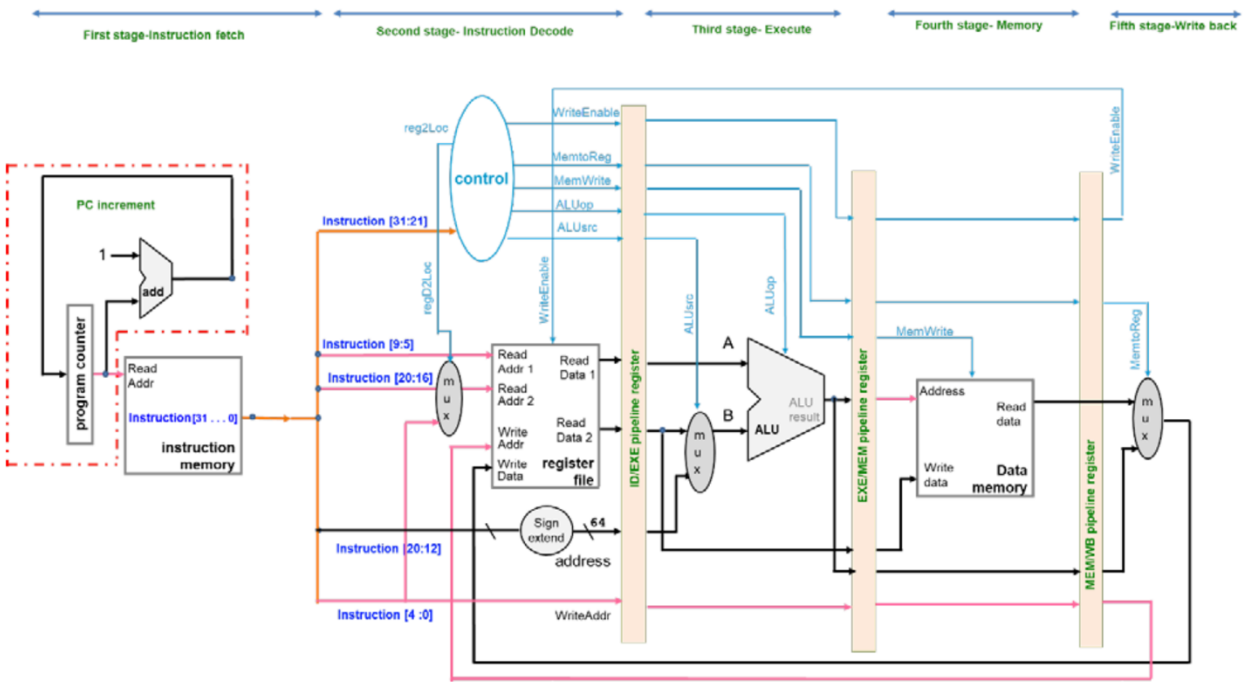

Fig. 3 Five-stage pipelined CPU (R and D type instructions).

We are having separate instruction memory and data memory. The data memory has 64-bit address port, a 64-bit input data port, a 64-bit output data port, and an active-high write-enable "MemWrite" signal. If the "MemWrite" signal is high, the memory will write the input data bits to the specified address (example for

STUR). The read operation is possible anytime for both instruction and data memories; once address is provided at its input port. The Verilog module for the data memory is given to you ("dmemory.v") it is almost similar to the instruction memory. The data is preloaded to the data memory. The data are loaded to data memory with the help of a text file by the name "dmem_test0.mem". Note that memory is clocked and hence we have a clock cycle delay for getting the output from memory.

D format instructions:
Example LDUR Rt, [Rn, #address], Meaning: [Rt] ← mem[[Rn] + sign ext(address)]: the content of registers Rn is added to the sign extended address to calculate the data memory address, and the value at that address location is written to destination register Rt. The bit assignments to different fields of D-format are shown below.

opcode	address	op	Rn	Rt
31 21	20 12	11 10	9 5	4

Examples:

1. LDUR X5, [X4, #4] (meaning: [X5] ← mem([X4] + 4)). The machine format is given below.

00000000101	000000100	00	00100	00101
31 21	20 12	11 10	9 5	4

2. STUR X8, [X9, #4] (meaning: [X8] → mem([X9] + 4)).

00000000110	000000100	00	01001	01000
31 21	20 12	11 10	9 5	4

We can note that register "Rt" acts as destination register for LDUR and source register for STUR. To facilitate this, we need a mux with a control signal "Reg2Loc" to select the source register for STUR as shown in Fig. 3. For LDUR and STUR instructions, the addresses calculated by ALU with the help of the "address" section of the instruction. The address section is sign extended to 64 bits in decode stage and is selected as one of the inputs in the execute stage. For this selection we need a multiplexer as shown in the execute stage whose select line is "alusrc" from control unit.

In the first stage, program counter 'PC.v' provides address to instruction memory. In the second stage, the instruction is read using "imemory.v" and the operand addresses are connected directly to the address ports of RF and the control logic. The instruction is decoded using "regfile.v" to generate the control signals using "control.v" and to get the data from the RF. The third stage comprises of execution using "ALU.v". In the fourth stage we have the data memory read and write using necessary control signals and final stage being written back. The ALU generates both data and addresses. For R-type instructions, the ALU calculates the data but for LDUR and STUR (D-type), ALU calculates the address. Finally, in the writeback stage, the ALU output (for R type) or the data memory output (for LDUR) is chosen with the help of control signal "memtoreg" and is written back to the register file.

Both IMEM and DMEM provide the output after one clock cycle. This provides an imaginative pipeline after instruction fetch. It is also the reason why the output of DMEM is not passed through the last pipeline stage (indicated in Fig.1).

The Verilog file "pipelined_five_stage.v" (provided) is the top module of the five-stage pipelined CPU. This top module consists of "PC.v", "regfile.v", "control.v", "imemory.v", "alu.v", "dmemory.v",

"ID_EXEstage.v", "EXE_MEMstage.v", and "MEM_WBstage.v". The "ID_EXEstage.v", "EXE_MEMstage.v", and "MEM_WBstage.v" are the pipeline registers between decode/execute, execute/datamemory and datamemory/writeback section respectively.

You can note that we are able to operate register type of instructions (ADD, SUB, AND, XOR, ORR) along with LDUR and STUR. In order to initialize the registers in the register file, data from the memory can be utilized.

1. We can note that "write enable" of register file is now no longer a constant as STUR do not write back to register. Hence "write enable" signal is a variable and is now pipelined through all stages.

2. Double click RTL schematic to view the circuit diagram for the five-stage pipelined implementation. The RTL schematic should closely resemble Fig. 3.

3. Test the 5-stage pipelined datapath implementation by using the test bench given in 'test_bench_5_stage_pipeline.v'.

4. As the instructions are already in IMEM (imem_test0.mem) and data in DMEM (dmem_test0.mem), we can simulate and see the functionality of the datapath. The IMEM uses file operation to get the instruction, and it reads "imem_test0.mem" as per the address given by PC. The DMEM reads from "dmem_test0.mem" as per the address given by ALU for LDUR. For STUR, the data from the register file (RF) is written to the DMEM using the address generated by ALU.

5. You can add new instructions and data by modifying the text files for both IMEM and DMEM. You can use https://personal.ntu.edu.sg/smitha/OPCoder/OPCoder/converter.html to convert the ARM instructions to machine code in binary or hexadecimal number system.

Example for verification of LDUR and STUR

1. In order to verify the operation, an example set of instructions and data given in Fig. 4 as "imem_test0.mem" and Fig. 5 as "dmem_test0.mem".

2. The set of instructions given in Fig. 4 has data dependencies. In order to get the correct result while running in a five-stage pipelined architecture; we need to add NOPs wherever necessary. Once the NOPS are correctly inserted to remove the data dependencies, we will get the added result in address location "[2]" of DMEM as can be seen in Fig. 6.

32 bit PC address (in hex)	32 bit instruction from IMEM	meaning
0000000000000000	001F03FF	ADD X31, X31, X31 (NOP)
0000000000000001	00A00045	LDUR X5, [X2,#0]
0000000000000002	00A02026	LDUR X6, [X1,#2]
0000000000000003	000600A5	ADD X5, X5, X6
0000000000000004	00C02045	STUR X5, [X2,#2]

Fig. 4 Instructions in the IMEM

64 bit address (in hex)	64 bit data from DMEM
0000000000000000	000000000000000A
0000000000000001	000000000000000A
0000000000000002	000000000000000A
0000000000000003	000000000000000A
0000000000000004	000000000000000A
0000000000000005	000000000000000A
0000000000000006	000000000000000A
0000000000000007	000000000000000A
0000000000000008	000000000000000A

Fig. 5 Data in the DMEM

3. Using STUR instruction, we can write back to DMEM register. Note that we are not writing back to the file "dmem_test0.mem", the update is only visible at the DMEM register as shown in Fig. 6. To see the content of dmemory data, you can use the Tcl Console to get access to its values, you may first run the following command:

```
get_objects /test_bench_5_stage_pipeline/uut/dm/*
```

You may see the path of "dmemory" array is listed as "/test_bench_5_stage_pipeline/uut/dm/dmemory", then you can obtain the values of the dmemory data by running the following commands **one-by-one**:

```
set NREG 64

set scope_path "/test_bench_5_stage_pipeline/uut/dm"

for {set i 0} {$i < $NREG} {incr i} {
    set dm_value [get_value ${scope_path}/dmemory[$i]]
    puts "dmemory[$i] = $dm_value"
}
```

X5 has the result of [A + A] = [14] (in hexadecimal), it is written to location 2 as Fig 6. If NOPs are not correctly inserted the result written back to DMEM will be wrong.

```
dmemory[0] = 000000000000000a
dmemory[1] = 000000000000000a
dmemory[2] = 0000000000000014
dmemory[3] = 000000000000000a
dmemory[4] = 000000000000000a
dmemory[5] = 000000000000000a
dmemory[6] = 000000000000000a
dmemory[7] = 000000000000000a
dmemory[8] = 000000000000000a
dmemory[9] = XXXXXXXXXXXXXXXX
dmemory[10] = XXXXXXXXXXXXXXXX
dmemory[11] = XXXXXXXXXXXXXXXX
dmemory[12] = XXXXXXXXXXXXXXXX
dmemory[13] = XXXXXXXXXXXXXXXX
dmemory[14] = XXXXXXXXXXXXXXXX
dmemory[15] = XXXXXXXXXXXXXXXX
```

Fig 6. DMEM register after executing the modified instructions in Fig. 4 for data in Fig. 5

EVALUATION - 2

2) Synthesize the five-stage pipelined CPU given in Fig. 3. Find the LUT consumption, number of registers and minimum clock period.

CPU	No of LUT slices	No of slice registers	Minimum Clock Period
Five stage pipelined CPU with LDUR and STUR			

Table 2: Slices and delay for five-stage pipelined CPU implementation.

(**Please make sure to implement all the experiments and fill out Table 2 in the last page of this manual. You'll need to submit the outcomes at the end of this lab**.)

Lab Report for Lab 3 (Name:_____, Matric:_____, Group:_____)

EVALUATION

1) Synthesize the four-stage pipelined CPU for R type instructions. Find the LUT consumption, number of registers and minimum clock period.

CPU	No of LUT slices	No of slice registers	Minimum Clock Period
Four stage pipeline			

Table 1: Slices and delay for four-stage pipelined CPU implementation.

2) Synthesize the five-stage pipelined CPU given in Fig.3. Find the LUT consumption, number of registers and minimum clock period.

CPU	No of LUT slices	No of slice registers	Minimum Clock Period
Five stage pipelined CPU with LDUR and STUR			

Table 2: Slices and delay for five-stage pipelined CPU implementation.