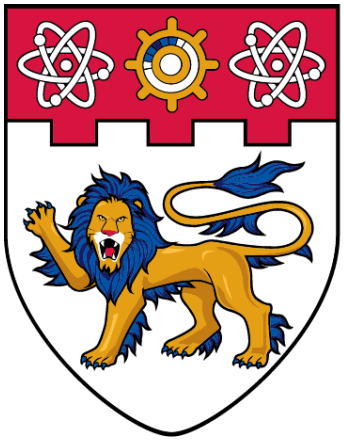


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

LAB 4:
FIVE-STAGE PIPELINE IMPLEMENTATION ON A MIPS PROCESSOR (32-BITS)

CZ3001 – ADVANCED COMPUTER ARCHITECTURE
(Semester 2, AY 2019/2020)

WILSON THURMAN TENG
U1820540H
SSP2

Introduction

$e = (a - b) * (c \text{ ^ } d)$
^ indicates XOR operation
* indicates multiply operation
- indicates subtraction operation

In this experiment, we implement a 5-Stage Pipeline (5SP) architecture for the MIPS 32 processor and evaluate its performance on the computation shown on the left.

Instruction Logic

PEMDAS Rule
P — Parentheses
E — Exponents
M — Multiplication
D — Division
A — Addition
S — Subtraction

In our implementation, we follow the PEMDAS rule and simplify the variables inside the parentheses first before any other computations are done.

1. Subtract *b* from *a*.
2. Perform XOR on variables *c* and *d*.
3. Multiply the results obtained from steps 1 and 2.

Approach

Initial values in Data Memory (DMEM)

0x00000001	a = 0xD
0x00000002	b = 0x3
0x00000003	c = 0xE
0x00000004	d = 0x2
0x00000005	e = 0xF

The approach to deriving the MIPS assembly code are as follows:

1) Use Load Word (LW) instruction to transfer values from DMEM addresses 0x01, 0x02, 0x03 and 0x04 into registers $1, $2, $3 and $4 respectively.
2) Use the SUB instruction to subtract the value in $2 from the value in $1 and store the result in $1. [a-b]
3) Use the XOR instruction between the values in $3 and $4 and store the result in $3. [c^d]
4) Use the MUL instruction to multiply the values in $1 and $3 and store the result into $5. [(a-b)*(c^d)]
5) Lastly, store the value in $5 into DMEM address 0x05.

PART A) MIPS Assembly Code

In order to compute the mathematical equation, register format (R-type) and immediate format (I-type) instructions will be required. The following are the minimum set of instructions needed.

Instruction	Hexadecimal Representation	Remarks
LW $1, 1($0)	1C010001	Load 'a' into register, $1
LW $2, 2($0)	1C020002	Load 'b' into register, $2
LW $3, 3($0)	1C030003	Load 'c' into register, $3
LW $4, 4($0)	1C040004	Load 'd' into register, $4
SUB $1, $1, $2	04220800	$1 <= $1 - $2
XOR $3, $3, $4	0C641800	$3 <= $3 ^ $4
MUL $5, $1, $3	14232800	$5 <= $1 * $3
SW $5, 5($0)	20050005	Store the result in register $5 to DMEM address 0x05

Minimum number of instructions = 8

PART B) Modified MIPS Assembly Code for Five-Stage Pipeline architecture

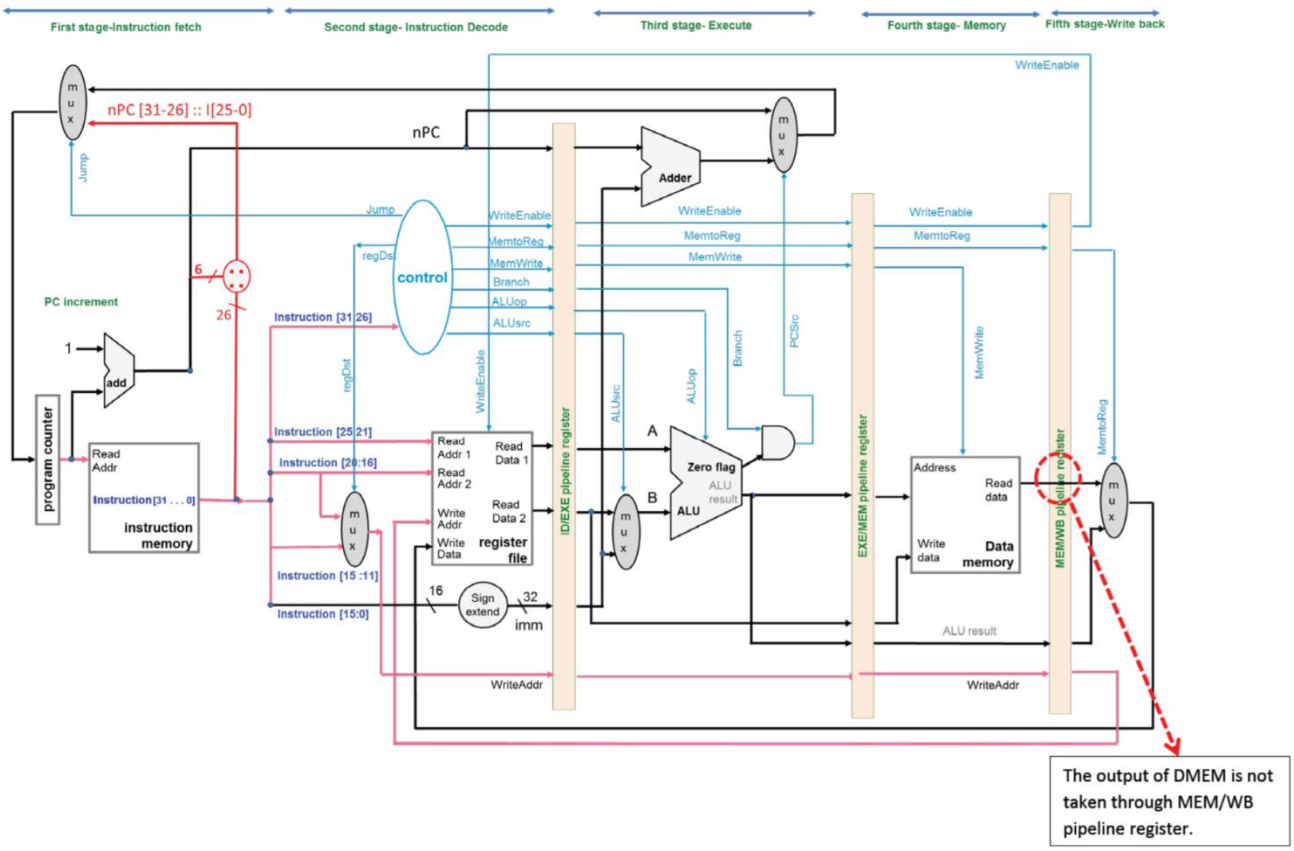


Fig. 1 Five-stage Pipelined CPU implementation diagram

In this implementation, we will use the Five-stage Pipelined CPU architecture as shown in *Fig.1* taken from the CZ3001 Advanced Computer Architecture lab 4 manual.

No.	Instruction	Clock Cycle																
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	LW $1, 1($0)	F	D	E	M	W												
2	LW $2, 2($0)		F	D	E	M	W											
3	LW $3, 3($0)			F	D	E	M	W										
4	LW $4, 4($0)				F	D	E	M	W									
5	SUB $1, $1, $2					F	D	E	M	W								
6	XOR $3, $3, $4						F	D	E	M	W							
7	MUL $5, $1, $3								F	D	E	M	W					
8	SW $5, 5($0)									F	D	E	M	W				

However, as depicted in the table above, the assembly code given in Part A would not be suitable for a Five-Stage Pipeline architecture. The cells shaded red, orange and yellow indicate the data hazard between the instructions.

All shaded cells are a form of data hazard known as the RAW (Read-After-Write) dependency. This dependency arises when one instruction relies on the output from a previous instruction. This will result in inaccurate results as the instruction will execute before the new value from previous instructions has been updated.

No.	Instruction	Clock Cycle																
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	LW $1, 1($0)	F	D	E	M	W												
2	LW $2, 2($0)		F	D	E	M	W											
3	LW $3, 3($0)			F	D	E	M	W										
4	LW $4, 4($0)				F	D	E	M	W									
5	SUB $1, $1, $2					F	D	E	M	W								
6	NOP						F	D	E	M	W							
7	XOR $3, $3, $4							F	D	E	M	W						
8	NOP								F	D	E	M	W					
9	NOP									F	D	E	M	W				
10	MUL $5, $1, $3										F	D	E	M	W			
11	NOP											F	D	E	M	W		
12	NOP												F	D	E	M	W	
13	SW $5, 5($0)													F	D	E	M	W

By inserting NOP instructions, we can remove these RAW dependencies. For example, in cycle 8, the value from register $1 and $2 would be updated in time for the SUB instruction (instruction 5) after the addition of a NOP instruction. Likewise, similar conclusions can be drawn for cycle 11 and 14.

Minimum number of instructions = 13

PART C) ISIM Simulation

Hardware Specifications:
Family : Spartan 6
Device : XC6SLX4
Package : CSG225
Speed : -3

Fig.2 Initial values in DMEM from addresses 0x01 to 0x04:

	0	1	2	3
0x0	00000000	0000000D	00000003	0000000E
0x4	00000002	0000000F	XXXXXXXX	XXXXXXXX

Fig.3 Instruction Memory (IMEM):

	0	1	2	3
0x0	1C010001	1C020002	1C030003	1C040004
0x4	04220800	00000000	0C641800	00000000
0x8	00000000	14232800	00000000	00000000
0xC	20050005	XXXXXXXX	XXXXXXXX	XXXXXXXX

Fig.4 Final values in DMEM from addresses 0x01 to 0x04 – after running assembly code in part B:

	0	1	2	3
0x0	00000000	0000000A	00000003	0000000C
0x4	00000002	00000078	00000000	00000000

This is in line with our calculations:
a – b = 0xD – 0x3 = 0xA (10)
c ^ d = 0xE ^ 0x2 = b'01110 ^ b'00010 = b'01100 = 0xC (12)
e = (a-b) * (c^d) = (0xA) * (0xC) = 0x78 (120)

PART D) Five-stage pipeline

Fig.5 Full Window (all values in Hexadecimal):

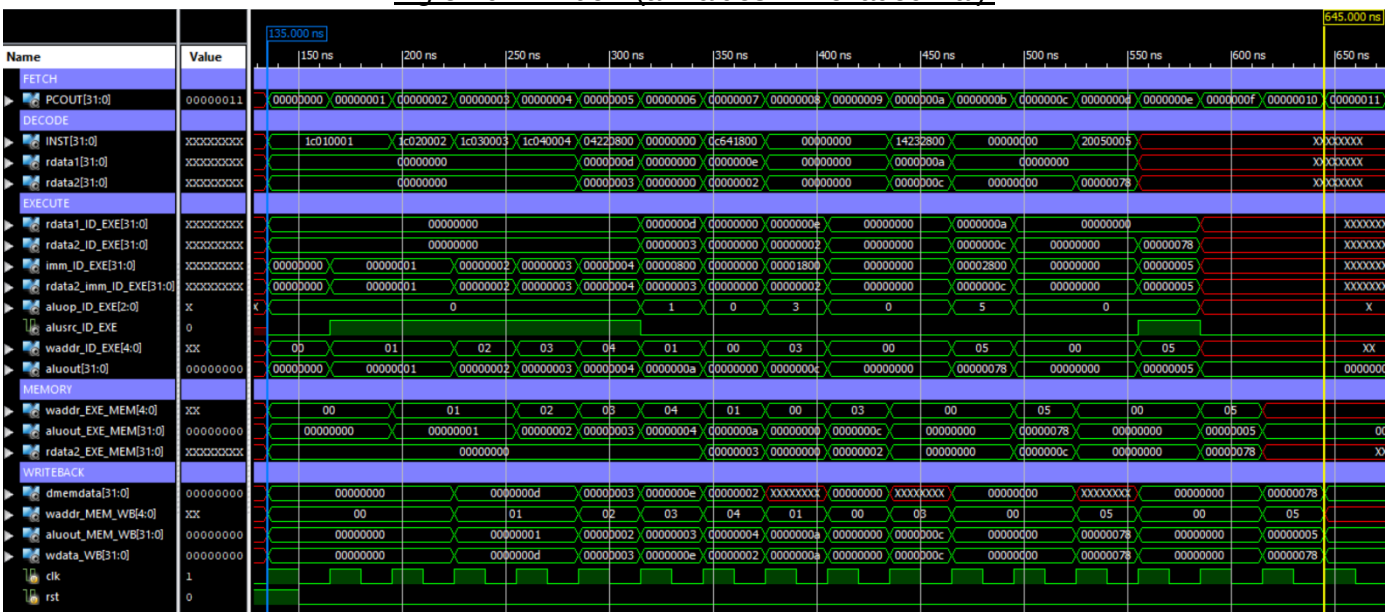
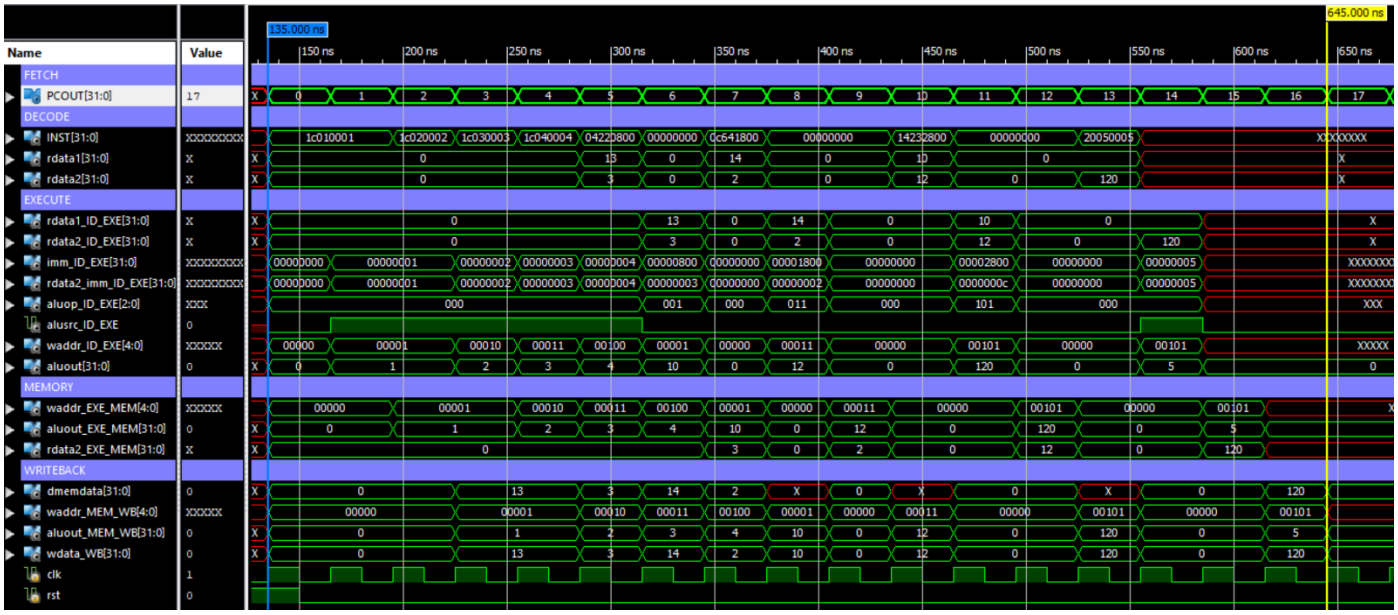


Fig.5 shows all 17 instruction cycles (0 to 16) and gives an overview of the processes that occur within the 5-stage pipeline.

Fig.6 Full Window (Changed some values to signed decimal):



In the *Fig.6*, instruction count, as well as data memory values and ALU outputs are changed to signed decimal representation. In subsequent sections of this report, this representation will be used. This allows for easier interpretation of the diagram.

Overview of the 5 Stages in an Immediate-type instruction

NOTE: The input/output names of the various components in the Five-Stage Pipeline architecture will be referenced from ***Fig.1*** for the remainder of this report.

1) **Fetch (instruction cycle 1, 0x00):**

This stage is where the program counter (PC) provides the Read Address to the IMEM. The corresponding instruction is then fetched.

2) **Decode (instruction cycle 2, 0x01):**

Different Instructions have different decoding procedures. In the case of an I-type instruction, the decoding is shown below.

Opcode inst [31:26]	**Src. Reg inst[25:21]**	**Dest. Reg inst[20:16]**	**Offset inst[15:0]**
first 6 bits	5 bits	5 bits	last 16 bits
Passed to ctrl unit to generate ctrl signals	Source Register	Destination Register	Immediate value before sign extension

The 6-bit opcode is then passed into the control unit (ctrl) to generate the 8 control signals shown below:

- **regDst**
- **WriteEnable**
- **MemToReg**
- **MemWrite**
- **Branch**
- **ALUop**
- **ALUsrc**
- **Jump**

regDst is used in this stage to determine whether the Destination Register address or instr[15:11] passes through the multiplexor as the **WriteAddr**. In the I-type instructions LW and SW, **regDst = 0**.

The 16-bit offset value will also sign extended to 32-bits.

The generated control signals, together with the other values are then stored in the **ID_EXE Pipeline Register** for use in the Execute stage.

3) **Execute (instruction cycle 3, 0x02):**

In the execute stage, 2 major processes occur, an arithmetic logical unit (ALU) operation is performed and the PC is updated to the branch location if both **branch** and the zero flag from the ALU equals to 1. In the case of LW and SW instructions, **Branch = 0**, so PC will be updated to the next instruction and branching will not occur.

The remaining control signals generated in the Decode Stage, together with the other values are then stored in the **EXE_MEM Pipeline Register** for use in the Memory Access stage.

4) **Memory Access (instruction cycle 4, 0x03):**

The Memory Access stage is where reads/writes to the DMEM occurs.

If **MemWrite = 0**, reading of an address in the DMEM occurs.
If **MemWrite = 1**, data will be written to the DMEM.

The remaining control signals generated in the Decode Stage, together with the other values are then stored in the **MEM_WB Pipeline Register** for use in the Writeback stage.

NOTE: The output of DMEM, **ReadData** will not be passed into the MEM_WB Pipeline Register. Instead, it is passed directly into the multiplexor in the next stage.

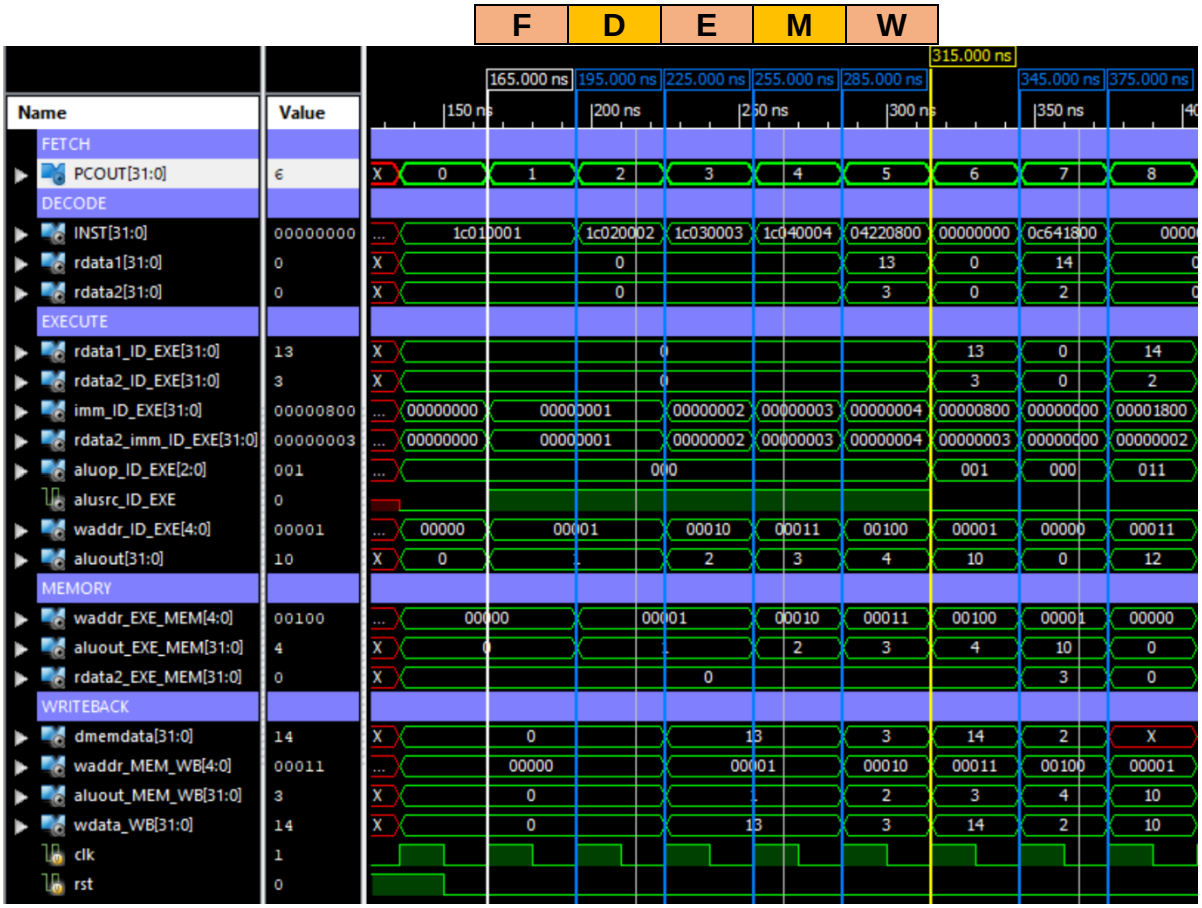
5) **Writeback (instruction cycle 5, 0x04):**

Provided **WriteEnable =** 1, the WriteBack stage allows data to be written back to the Register File.

The data to be written is obtained either directly from the DMEM's **ReadData** output or the ALUresult stored in the MEM_WB Pipeline Register, depending on the control signal **MemToReg** passed into the multiplexor.

The **WriteAddr** passed in the Register File will the value stored in the MEM_WB Pipeline Register (generated in the Decode Stage and passed through Pipeline Registers)

Load Word (LW) Instruction:



To demonstrate how the I-type base addressing instruction LW is executed in a Five-Stage Pipeline, the first instruction **LW $1, 1($0)** will be analysed.

1) **Fetch (instruction cycle 1, 0x00):**

The program counter (PC) provides the Read Address, 0x0000_0000 to the IMEM. The instruction 0x1C010001 is then fetched.

2) **Decode (instruction cycle 2, 0x01):**

Since LW is an I-type instruction, it follows the decoding procedure shown below:

LW $1, 1($0) = 0x1C010001 = b'0001_1100_0000_0001_0000_0000_0000_0001			
Opcode inst [31:26]	**Src. Reg inst[25:21]**	**Dest. Reg inst[20:16]**	**Offset inst[15:0]**
first 6 bits	5 bits	5 bits	last 16 bits
b'00_0111	b'0_0000	b'0_0001	b'0000_0000_0000_0001
Passed to ctrl unit	Register $0	Register $1	immediate value before sign extension

When the 6-bit opcode is passed into the control unit (ctrl), the following 8 control signals are generated:

regDst = 0 | WriteEnable = 1 | MemToReg = 0 | MemWrite = 0 | Branch = 0 | ALUop = 000 | ALUsrc = 1 | Jump = 0 |

As shown in the table above, the 5-bit source register address and the 5-bit destination register address, b'0_0000 and b'0_0001 respectively, are derived from the next 10 bits.

The source register address of b'0_0000 is passed into the Register file as **ReadAddr1**, and the Register File outputs the corresponding 32-bit data of 0x0000_0000 in register $0 as **ReadData1**.

The destination address is passed into the Multiplexor together with the instr[15:11] (no meaning in this situation). Since **regDst = 0**, the source destination address is passed and stored in the ID_EXE Pipeline Register as **WriteAddr**.

The 16-bit offset value of 0x0001 is then sign extended to 32-bits to obtain an offset of 0x0000_0001 (Immediate value = 1). This 32-bits value of 1 is regarded as **imm**.

The following values of interest are then stored in the **ID_EXE Pipeline Register** for use in the Execute stage:

ReadData1 = 0x0000_0000 | **WriteAddr = b'0_0001** | **Imm = 0x0000_0001** | **WriteEnable = 1 | MemToReg = 0 | MemWrite = 0 | Branch = 0 | ALUop = 000 | ALUsrc = 1 | Jump = 0 |**

3) **Execute (instruction cycle 3, 0x02):**

ReadData1 value of 0x0000_0000 is passed into the ALU as the first operand.

Imm is passed into the multiplexor and since **ALUsrc = 1**, the 32-bit **Imm** value of 0x0000_00001 is passed into the ALU as the second operand.

Since **ALUop = b'000**, The ALU performs an ADD operation on the 2 operands:

ALUresult = ReadData1 + Imm = 0x0000_0001

The following values of interest are then stored in the **EXE_MEM Pipeline Register** for use in the Memory Access stage:

ALUresult = 0x0000_0001 | **WriteAddr = b'0_0001** | **WriteEnable = 1 | MemToReg = 0 | MemWrite = 0 |**

4) **Memory Access (instruction cycle 4, 0x03):**

Since **MemWrite = 0**, reading of an address in the DMEM occurs. **ALUresult**'s address of 0x0000_0001 is passed into the DMEM as **Address**, and the corresponding content of 0x0000_000D is fetched as **ReadData**.

The following values of interest are then stored in the **MEM_WB Pipeline Register** for use in the Writeback stage:

WriteAddr = b'0_0001 |WriteEnable = 1 |MemToReg = 0 |

NOTE: **ReadData** of value 0x0000_000D is not passed into the MEM_WB Pipeline Register but passed directly into the multiplexor in the next stage.

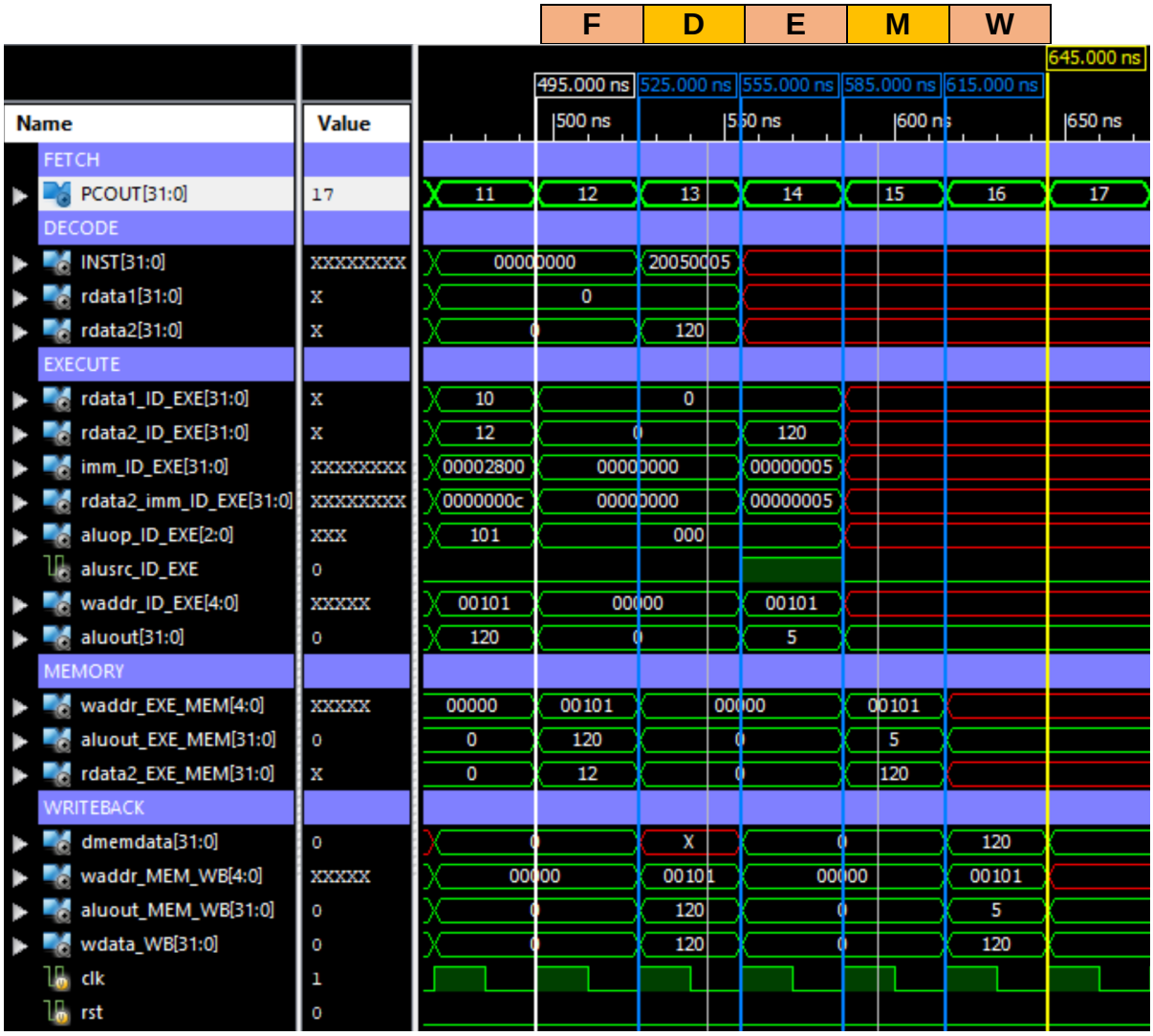
5) **Writeback (instruction cycle 5, 0x04):**

The **WriteAddr = b'0_0001** stored in the MEM_WB Pipeline Register is passed into the Register File as **WriteAddr**.

Since **MemToReg = 0**, the multiplexor will pass the **ReadData** value of 0x0000_000D obtained directly from the DMEM into the Register File as **WriteData**.

Coupled with **WriteEnable = 1**, the value of 0x0000_0000D will be written to Register $1 of the Register File.

Store Word (SW) Instruction:



To demonstrate how the I-type base addressing instruction SW is executed in a Five-Stage Pipeline, the first instruction **SW $5, 5($0)** will be analysed.

1) **Fetch (instruction cycle 13, 0x0C):**

The program counter (PC) provides the Read Address, 0x0000_000C to the IMEM. The instruction 0x20050005 is then fetched.

2) **Decode (instruction cycle 14, 0x0D):**

Since SW is an I-type instruction, it follows the decoding procedure shown below:

SW $5, 5($0) = 0x20050005 = b'0010_0000_0000_0101_0000_0000_0000_0101			
Opcode inst [31:26]	**Src. Reg inst[25:21]**	**Dest. Reg inst[20:16]**	**Offset inst[15:0]**
first 6 bits	5 bits	5 bits	last 16 bits
b'00_1000	b'0_0000	b'0_0101	b'0000_0000_0000_0101
Passed to ctrl unit	Register $0	Register $5	immediate value before sign extension

When the 6-bit opcode is passed into the control unit (ctrl), the following 8 control signals are generated:

regDst = 0 | **WriteEnable = 0** | **MemToReg = 0** | **MemWrite = 1** | **Branch = 0** | **ALUop = 000** | **ALUsrc = 1** | **Jump = 0 |**

As shown in the table above, the 5-bit source register address and the 5-bit destination register address, b'0_0000 and b'0_0101 respectively, are derived from the next 10 bits.

The Source Register address of b'0_0000 is passed into the Register file as **ReadAddr1**, and the Register File outputs the corresponding 32-bit data of 0x0000_0000 in register $0 as **ReadData1**.

The Destination Register address of b'0_00101 is passed into the Register file as **ReadAddr2**, and the Register File outputs the corresponding 32-bit data of 0x0000_0078 in register $5 as **ReadData2**.

The 16-bit offset value of 0x0101 is then sign extended to 32-bits to obtain an offset of 0x0000_0101 (Immediate value = 5). This 32-bits value of 5 is regarded as **imm**.

The following values of interest are then stored in the **ID_EXE Pipeline Register** for use in the Execute stage:

ReadData1 = 0x0000_0000 | **ReadData2 = 0x0000_0078** | **Imm = 0x0000_0101** | **WriteEnable = 0** | **MemToReg = 0** | **MemWrite = 1** | **Branch = 0** | **ALUop = 000** | **ALUsrc = 1** |

3) **Execute (instruction cycle 15, 0x0E):**

ReadData1 value of 0x0000_0000 is passed into the ALU as the first operand.

Imm is passed into the multiplexor and since **ALUsrc = 1**, the 32-bit **Imm** value of 0x0000_00101 is passed into the ALU as the second operand.

Since **ALUop = b'000**, The ALU performs an ADD operation on the 2 operands:

ALUresult = ReadData1 + Imm = 0x0000_0101

The following values of interest are then stored in the **EXE_MEM Pipeline Register** for use in the Memory Access stage:

ALUresult = 0x0000_0101 | **ReadData2 = 0x0000_0078** | **WriteEnable = 0** | **MemToReg = 0** | **MemWrite = 1 |**

4) **Memory Access (instruction cycle 16, 0x0F):**

Since **MemWrite = 1**, data will be written to the DMEM.

ALUresult's address of 0x0000_0101 is passed into the DMEM as **Address**.

ReadData2's value of 0x0000_0078 stored in the EXE_MEM Pipeline Register is passed into the DMEM as **WriteData**.

Therefore, the value of 0x0000_0078 is now stored in address 0x0000_0101 of the DMEM.

The following values of interest are then stored in the **MEM_WB Pipeline Register** for use in the Writeback stage:

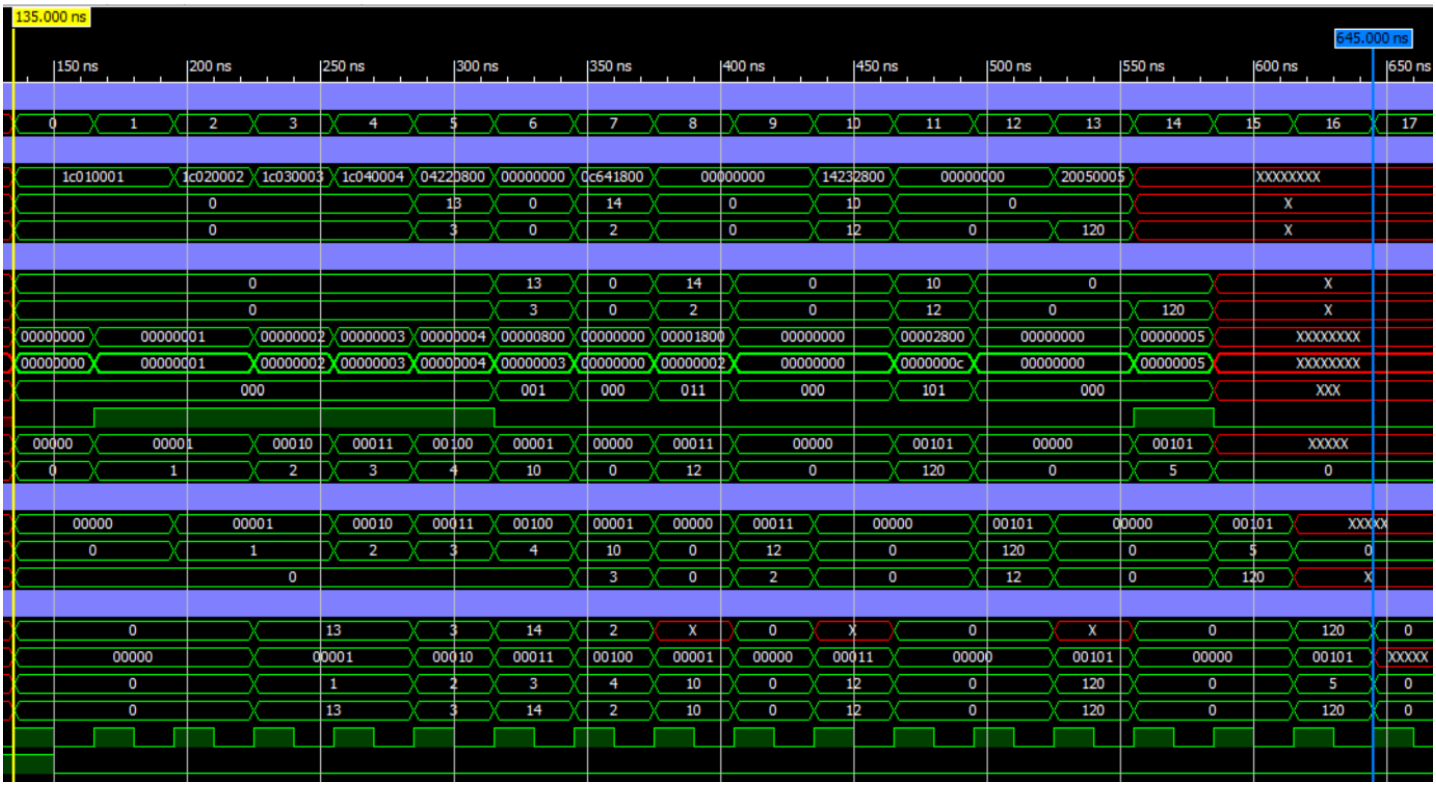
| **WriteEnable = 0** | **MemToReg = 0** |

5) **Writeback (instruction cycle 17, 0x10):**

Since **WriteEnable = 0**, writes to the Register File are disallowed and hence, the Register File remains unchanged.

Although there is no WriteBack stage for the SW instruction, this extra clock cycle is still required for the Five-Stage Pipeline architecture to operate without errors.

PART E) Execution Time



Starting Time : 135.000ns
Ending Time : 645.000ns

Therefore, execution time is (645-135) = 510ns

PART F) Steady-State CPI

Steady State CPI formula:

$$Steady\ state\ CPI = \frac{No.of\ Instructions + No.of\ Stalls}{No.of\ Instructions}$$

Calculation for five-stage pipeline:

$$Steady\ state\ CPI = \frac{8+5}{8} = 1.625$$