|  |
| --- |
| 台科大 |
| PA3 |
| 30頁真的寫不完 |

|  |
| --- |
| 周 柏宇  2024/5/20 |

內容

[Part 1 1](#_Toc167312212)

[Adder 1](#_Toc167312213)

[Description 1](#_Toc167312214)

[IO 1](#_Toc167312215)

[Explanation 2](#_Toc167312216)

[ALU\_Control 2](#_Toc167312217)

[Description 2](#_Toc167312218)

[IO 2](#_Toc167312219)

[Explanation 2](#_Toc167312220)

[ALU 2](#_Toc167312221)

[Description 2](#_Toc167312222)

[IO 2](#_Toc167312223)

[Explanation 3](#_Toc167312224)

[Control 3](#_Toc167312225)

[Description 3](#_Toc167312226)

[IO 3](#_Toc167312227)

[Explanation 4](#_Toc167312228)

[DM 4](#_Toc167312229)

[Description 4](#_Toc167312230)

[IO 4](#_Toc167312231)

[Explanation 4](#_Toc167312232)

[EX\_MEM 5](#_Toc167312233)

[Description 5](#_Toc167312234)

[IO 5](#_Toc167312235)

[Explanation 5](#_Toc167312236)

[GPRMUX 6](#_Toc167312237)

[Description 6](#_Toc167312238)

[IO 6](#_Toc167312239)

[Explanation 6](#_Toc167312240)

[ID\_EX 6](#_Toc167312241)

[Description 6](#_Toc167312242)

[IO 6](#_Toc167312243)

[Explanation 7](#_Toc167312244)

[IF\_ID 8](#_Toc167312245)

[Description 8](#_Toc167312246)

[IO 8](#_Toc167312247)

[Explanation 8](#_Toc167312248)

[IM 8](#_Toc167312249)

[Description 8](#_Toc167312250)

[IO 8](#_Toc167312251)

[Explanation 8](#_Toc167312252)

[MEM\_WB 8](#_Toc167312253)

[Description 8](#_Toc167312254)

[IO 8](#_Toc167312255)

[Explanation 9](#_Toc167312256)

[R\_PipelineCPU 9](#_Toc167312257)

[Description 10](#_Toc167312258)

[IO 10](#_Toc167312259)

[Explanation 10](#_Toc167312260)

[RegMUX 11](#_Toc167312261)

[Description 11](#_Toc167312262)

[IO 11](#_Toc167312263)

[Explanation 11](#_Toc167312264)

[RF 11](#_Toc167312265)

[Description 11](#_Toc167312266)

[IO 11](#_Toc167312267)

[Explanation 11](#_Toc167312268)

[Part 2 12](#_Toc167312269)

[Changes 12](#_Toc167312270)

[Part 3 12](#_Toc167312271)

[Changes 12](#_Toc167312272)

[New modules 12](#_Toc167312273)

[FinalCPU 12](#_Toc167312274)

[Forwarding\_Unit 14](#_Toc167312275)

[HD\_Unit 15](#_Toc167312276)

[Stall\_MUX 15](#_Toc167312277)

[TMUX 16](#_Toc167312278)

[Part 1 Test Program 16](#_Toc167312279)

[RF 16](#_Toc167312280)

[Part 2 Test Program 16](#_Toc167312281)

[DM 16](#_Toc167312282)

[RF 17](#_Toc167312283)

[Part 3 Test Program 17](#_Toc167312284)

[DM 17](#_Toc167312285)

[RF 17](#_Toc167312286)

[Simulation Result 17](#_Toc167312287)

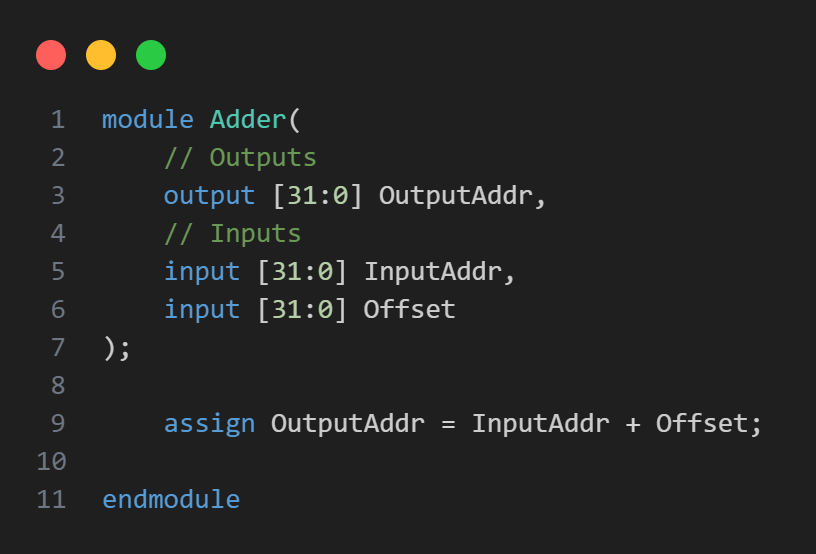
[Timing 17](#_Toc167312288)

[Utilization 17](#_Toc167312289)

[Power 17](#_Toc167312290)

# Part 1

## Adder



### Description

The purpose of the Adder module isto compute the sum of two 32-bit input values, InputAddr and Offset, and provide the result as a 32-bit output value, OutputAddr.

### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| InputAddr | 31 bits | Original pc addr. |
| Offset | 31 bits | How many to be added. |

Outputs:

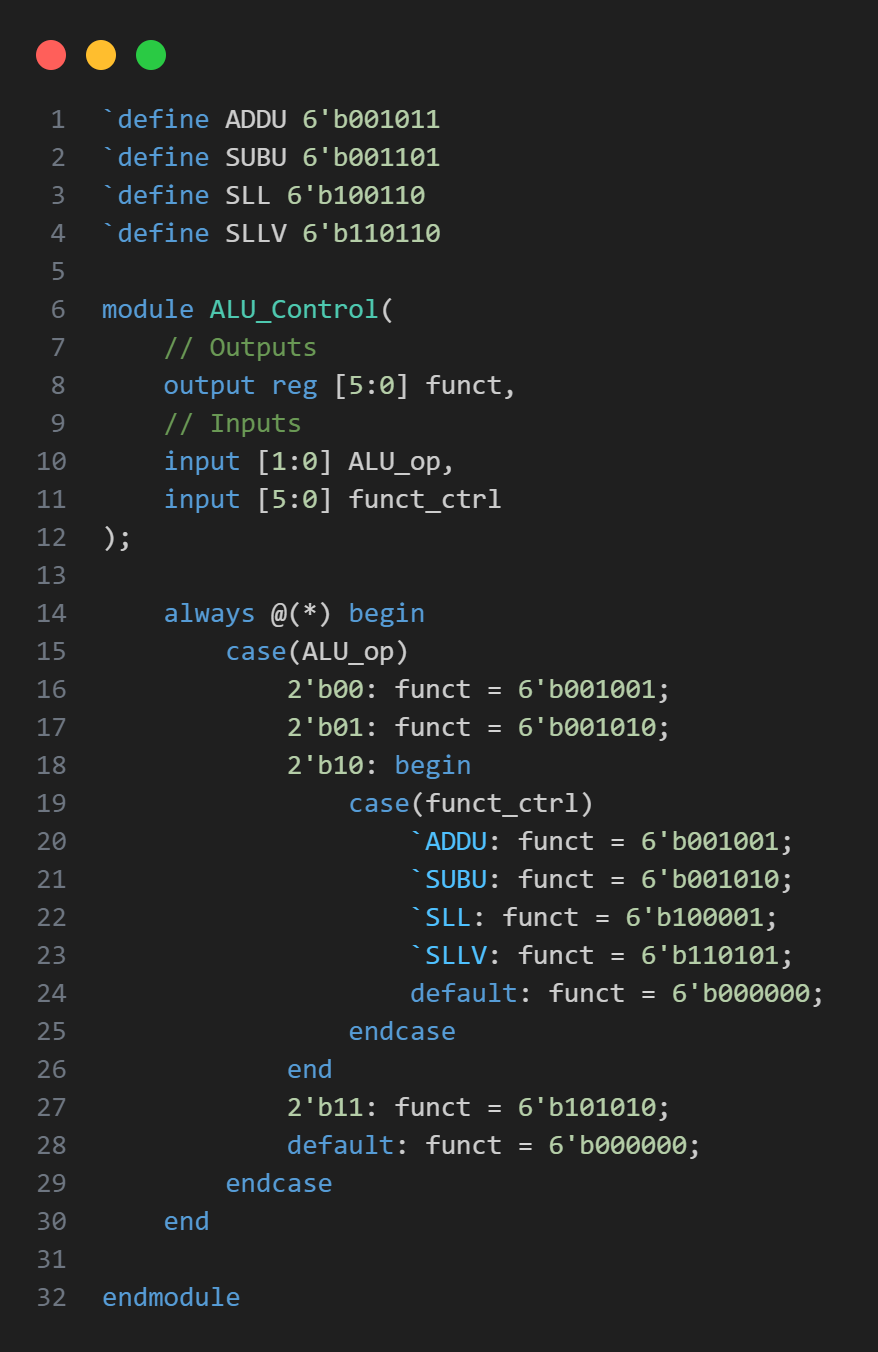
|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| OutputAddr | 31 bits | New pc addr. |

### Explanation

line 9

Using assign statement to perform a continuous addition of the inputs in.

## ALU\_Control



### Description

Generates a function code (**funct**) based on the ALU operation code (**ALU\_op**) and a function control code (**funct\_ctrl**).

### IO

Input:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| ALU\_op | 2 bits | ALU operation code that determines the type of operation. |
| funct\_ctrl | 6 bits | Function control code used for specific operations when **ALU\_op** is **2'b10**. |

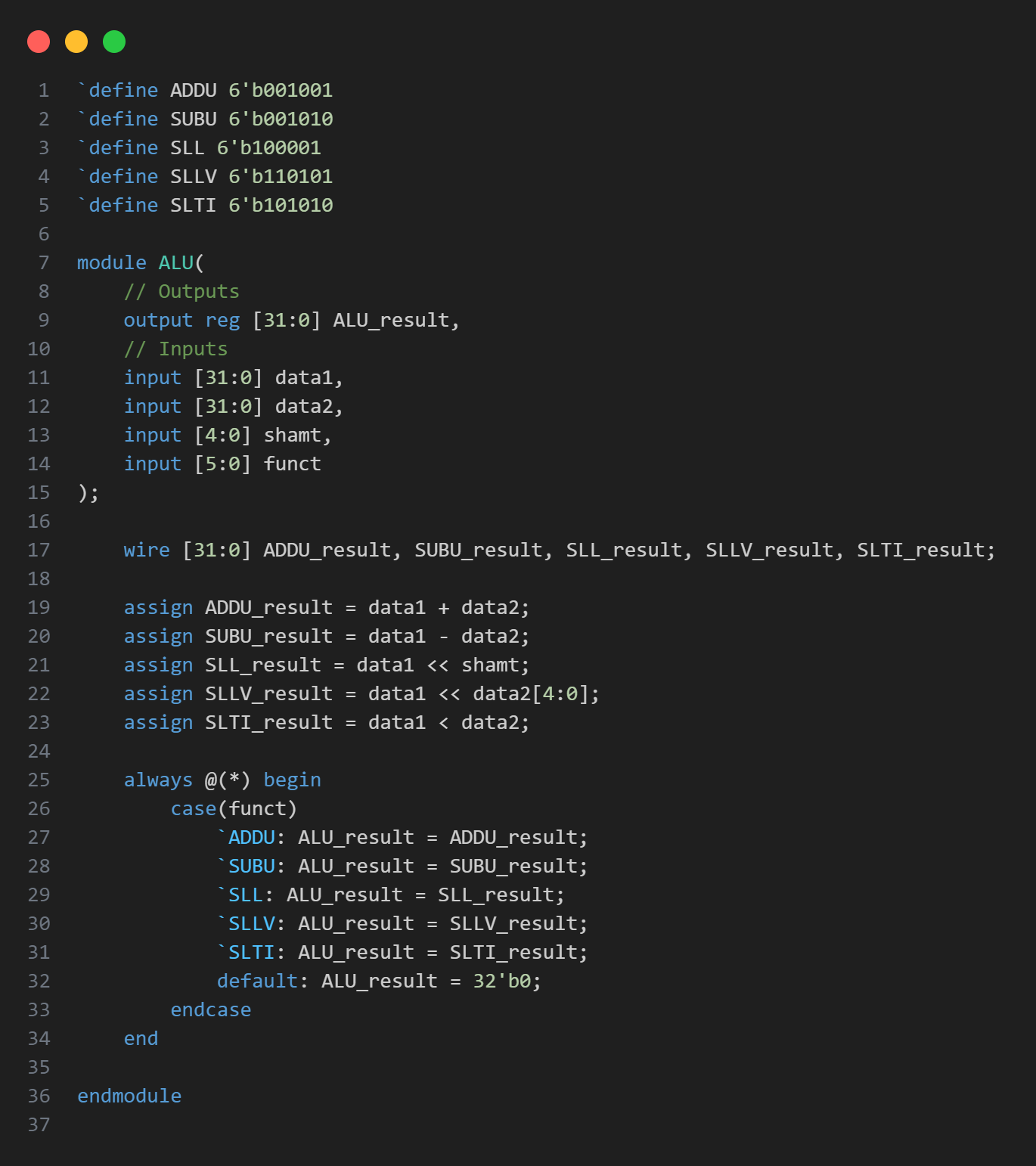
Outputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| funct | 6 bits | Function code output used to control the ALU operation. |

### Explanation

* When **ALU\_op** is **2'b00**, **funct** is set to **6'b001001**.
* When **ALU\_op** is **2'b01**, **funct** is set to **6'b001010**.
* When **ALU\_op** is **2'b11**, **funct** is set to **6'b101010**.
* When **ALU\_op** is **2'b10**
  + ADDU: funct is set to 6'b001001.
  + SUBU: funct is set to 6'b001010.
  + SLL: funct is set to 6'b100001.
  + SLLV: funct is set to 6'b110101.
  + For any other funct\_ctrl value, funct defaults to 6'b000000.

## ALU



### Description

Performs arithmetic and logical operations based on the function code (**funct**).

### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| data1 | 32 bits | First operand for the ALU operation. |
| data2 | 32 bits | Second operand for the ALU operation or shift amount in SLLV. |
| shamt | 5 bits | Shift amount for the shift left logical (SLL) operation. |
| funct | 6 bits | Function code that determines the specific ALU operation to perform. |

Outputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| ALU\_result | 32 bits | Result of the ALU operation. |

### Explanation

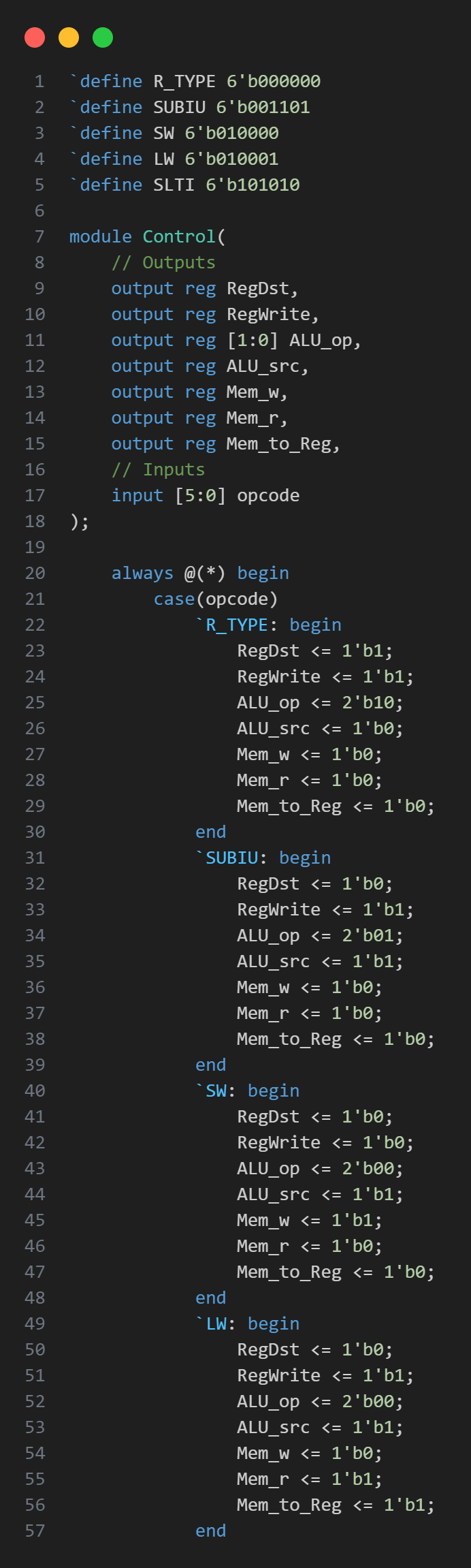
Line 19~23

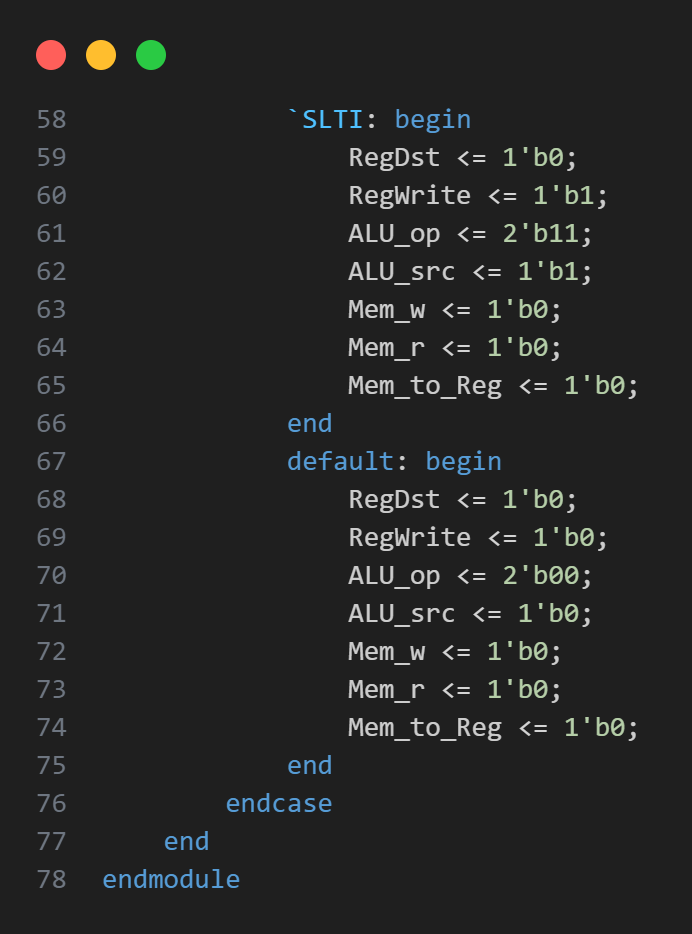
Using wire to get all result at the same time

Line 25~33

Put the value to ALU\_result designated by funct.

## Control





### Description

Generates control signals based on the opcode of an instruction.

### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| opcode | 6 bits | Opcode of the instruction to be decoded. |

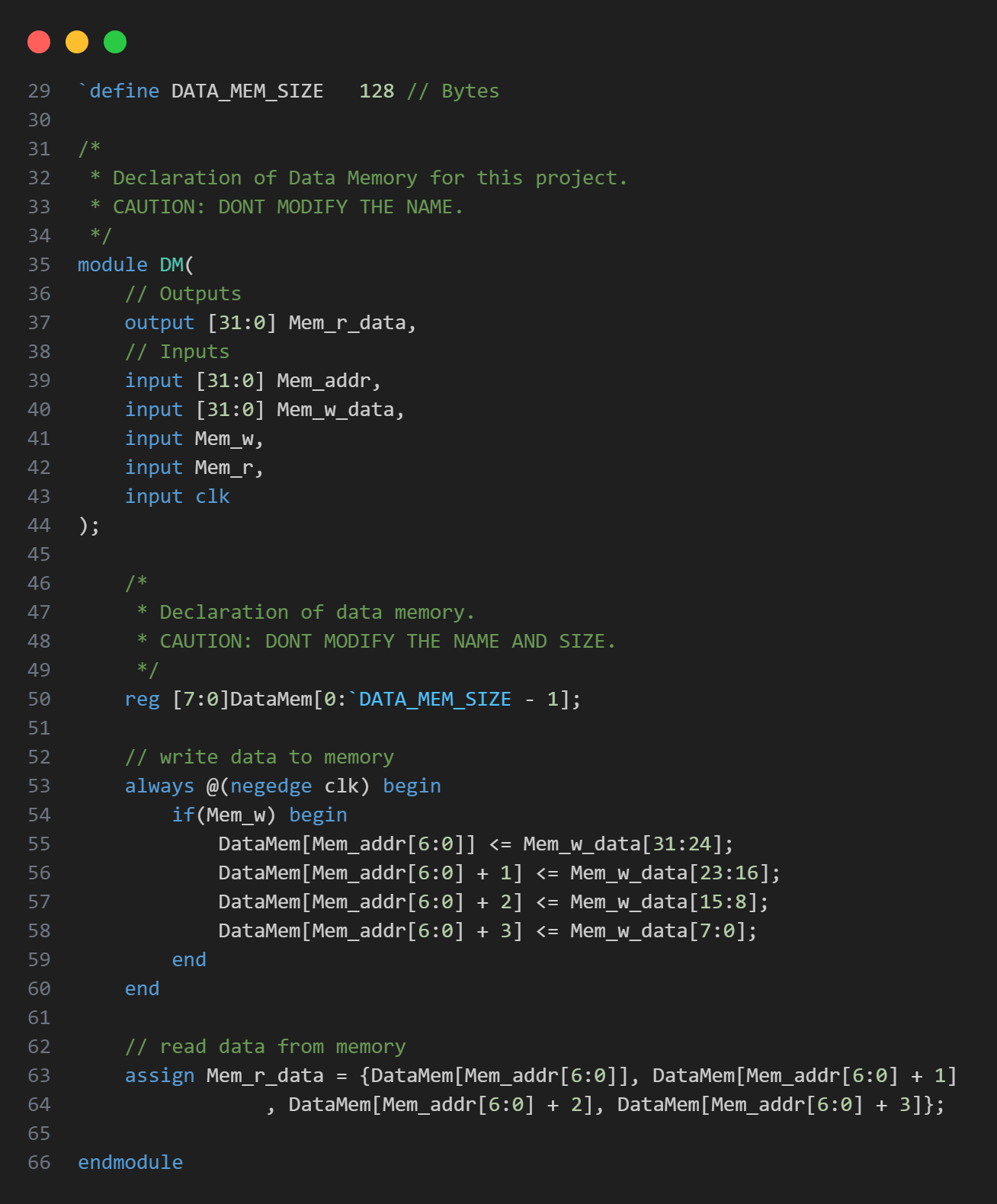
Outputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| RegDst | 1 bit | Determines the destination register for the write operation. |
| RegWrite | 1 bit | Enables writing to the register file. |
| ALU\_op | 2 bits | Selects the ALU operation to be performed. |
| ALU\_src | 1 bit | Selects the second operand for the ALU. |
| Mem\_w | 1 bit | Enables writing to memory. |
| Mem\_r | 1 bit | Enables reading from memory. |
| Mem\_to\_Reg | 1 bit | Selects the data source for writing to the register file. |

### Explanation

uses an **always @(\*)** block to generate control signals based on the input **opcode**.

## DM



### Description

Simulate a data memory component that allows reading from and writing to memory locations.

Not used in R type.

### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| Mem\_addr | 32 bits | Address from which to read or write data. |
| Mem\_w\_data | 32 bits | Data to be written to the memory. |
| Mem\_w | 1 bit | Write enable signal; when high, data is written to memory. |
| Mem\_r | 1 bit | Read enable signal. Not used in this implementation |
| clk | 1 bit | Clock signal. Not used in this implementation. |

Outputs:

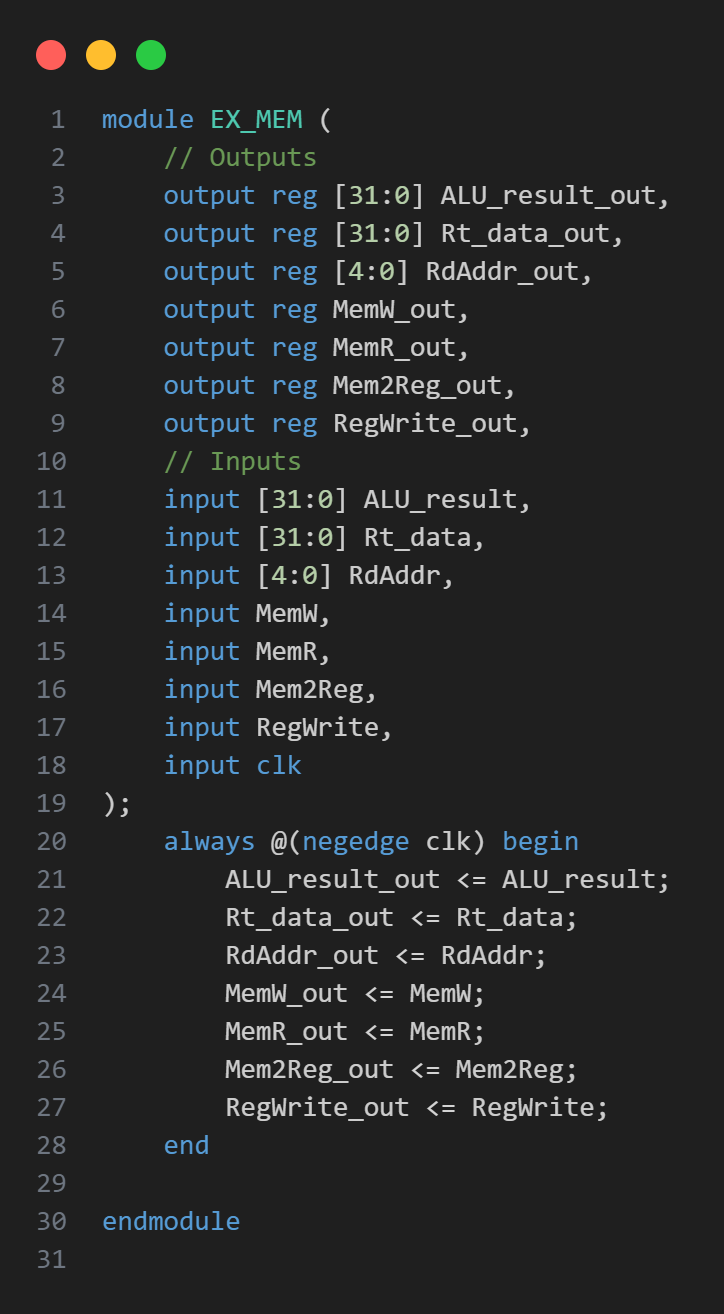
|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| Mem\_r\_data | 32 bits | Data read from the memory. |

### Explanation

Using **always @(\*)** block ensures that writing occurs whenever **Mem\_w** is high.

Using **assign** statement continuously updates **Mem\_r\_data** based on the current memory contents at the specified address.

## EX\_MEM



### Description

Stores the outputs from the execution stage and passes them to the memory stage in a pipelined processor.

### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| ALU\_result | 32 bits | Result of the ALU operation from the execution stage. |
| Rt\_data | 32 bits | Data from register **Rt** from the execution stage. |
| RdAddr | 5 bits | Register destination address from the execution stage. |
| MemW | 1 bit | Signal indicating memory write operation. |
| MemR | 1 bit | Signal indicating memory read operation. |
| Mem2Reg | 1 bit | Signal indicating whether memory data should be written back to a register. |
| RegWrite | 1 bit | Signal enabling register write operation. |
| clk | 1 bit | Clock signal. |

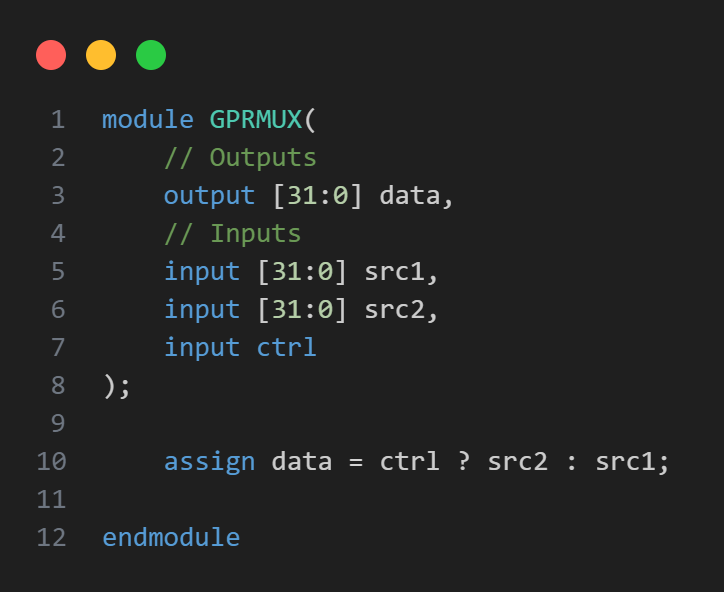
Outputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| ALU\_result\_out | 32 bits | Result of the ALU operation. |
| Rt\_data\_out | 32 bits | Data from register **Rt**. |
| RdAddr\_out | 5 bits | Register destination address for write-back. |
| MemW\_out | 1 bit | Signal indicating memory write operation. |
| MemR\_out | 1 bit | Signal indicating memory read operation. |
| Mem2Reg\_out | 1 bit | Signal indicating whether memory data is written back to a register. |
| RegWrite\_out | 1 bit | Signal enabling register write operation. |

### Explanation

Use registers to store the results and control signals from the EX stage on the falling edge of the clock.

## GPRMUX



### Description

2-to-1 multiplexer that selects one of two 32-bit input data signals based on a control signal.

### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| src1 | 32 bits | First input data. |
| src2 | 32 bits | Second input data. |
| ctrl | 1 bit | Control signal to select between **src1** and **src2**. |

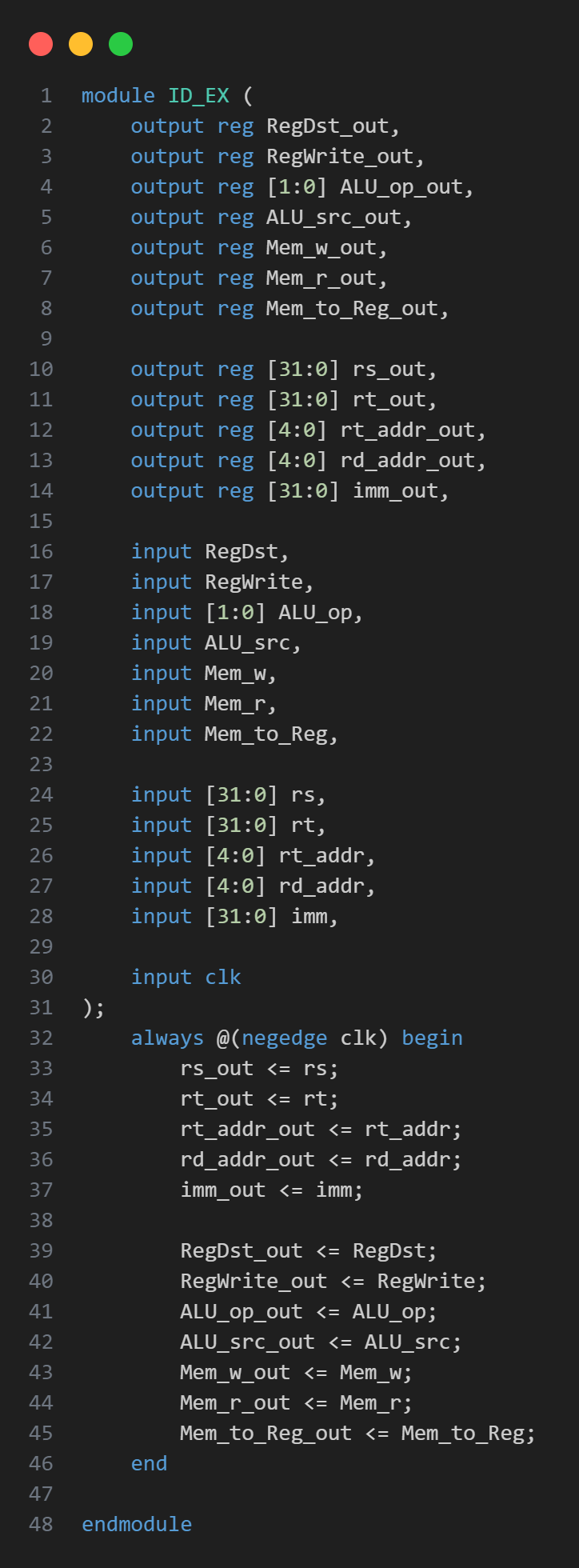
Outputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| data | 32 bits | Selected output data based on control signal. |

### Explanation

* When **ctrl** is **0**, the output **data** is equal to **src1**.
* When **ctrl** is **1**, the output **data** is equal to **src2**.

## ID\_EX



### Description

Stores intermediate values and control signals between the Instruction Decode (ID) stage and the Execution (EX) stage.

### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| RegDst | 1 bit | Control signal selecting the destination |
| RegWrite | 1 bit | Control signal enabling the register write operation. |
| ALU\_op | 2 bits | Control signals for ALU operation type. |
| ALU\_src | 1 bit | Control signal selecting the ALU source operand. |
| Mem\_w | 1 bit | Control signal enabling memory write operation. |
| Mem\_r | 1 bit | Control signal enabling memory read operation. |
| Mem\_to\_Reg | 1 bit | Control signal for selecting data source to write back to the register. |
| rs | 32 bits | Data from source register **rs**. |
| rt | 32 bits | Data from source register **rt**. |
| rt\_addr | 5 bits | Address of the **rt** register. |
| rd\_addr | 5 bits | Address of the **rd** register. |
| imm | 32 bits | Immediate value. |
| clk | 1 bit | Clock signal. |

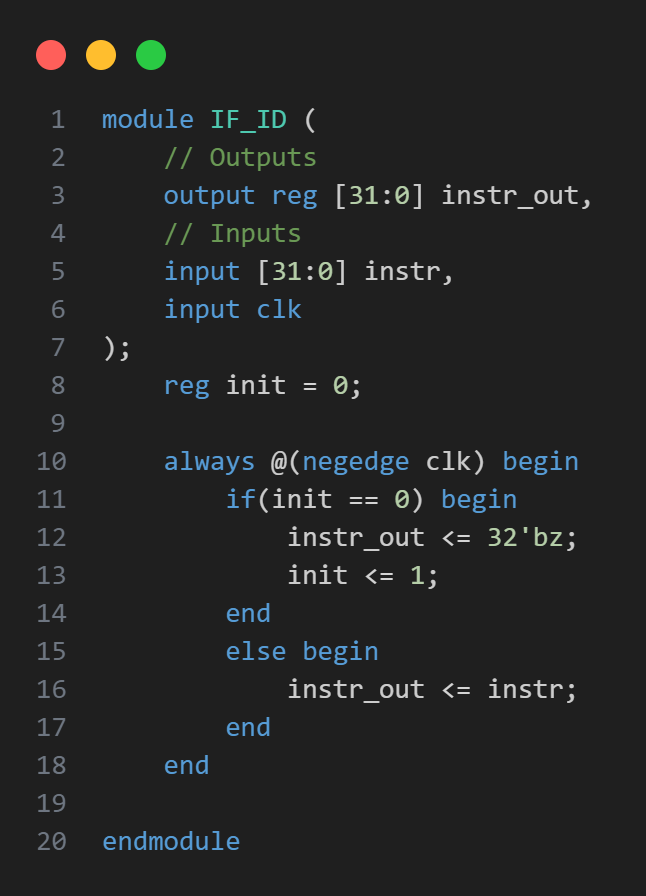
Outputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| RegDst\_out | 1 bit | Output control signal selecting the desination. |
| RegWrite\_out | 1 bit | Output control signal enabling the register write operation. |
| ALU\_op\_out | 2 bits | Output control signals for ALU operation type. |
| ALU\_src\_out | 1 bit | Output control signal selecting the ALU source operand. |
| Mem\_w\_out | 1 bit | Output control signal enabling memory write operation. |
| Mem\_r\_out | 1 bit | Output control signal enabling memory read operation. |
| Mem\_to\_Reg\_out | 1 bit | Output control signal for selecting data source to write back to the register. |
| rs\_out | 32 bits | Output data from source register **rs**. |
| rt\_out | 32 bits | Output data from source register **rt**. |
| rt\_addr\_out | 5 bits | Output address of the **rt** register. |
| rd\_addr\_out | 5 bits | Output address of the **rd** register. |
| imm\_out | 32 bits | Output immediate value. |

### Explanation

On the falling edge of the clock, the module transfers the input data and control signals to their respective output ports.

## IF\_ID



### Description

Captures and holds the instruction fetched from the instruction memory in the Instruction Fetch (IF) stage.

### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| instr | 32 bits | Input instruction from the instruction memory. |
| clk | 1 bit | Clock signal. |

Outputs:

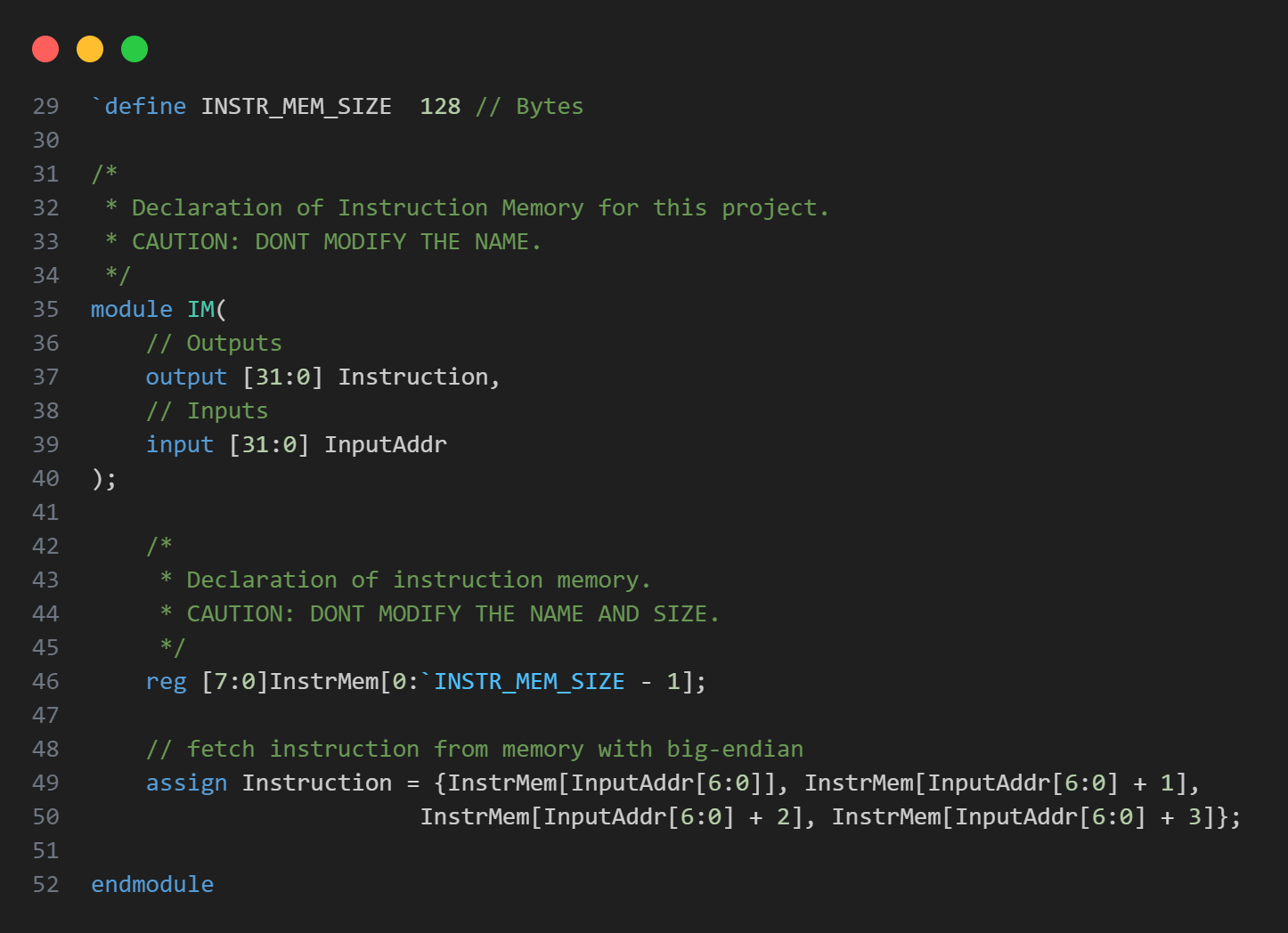
|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| instr\_out | 32 bits | Output instruction to be sent to the ID stage. |

### Explanation

Captures the **instr** input on the falling edge of the clock and updates the **instr\_out** output with this value.

Using init reg to manually deal with **WRONG** input for the specific case in this PA.

## IM



### Description

Fetches 32-bit instructions from memory based on a given input address.

### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| InputAddr | 32 bits | Input address used to fetch the instruction from memory. |

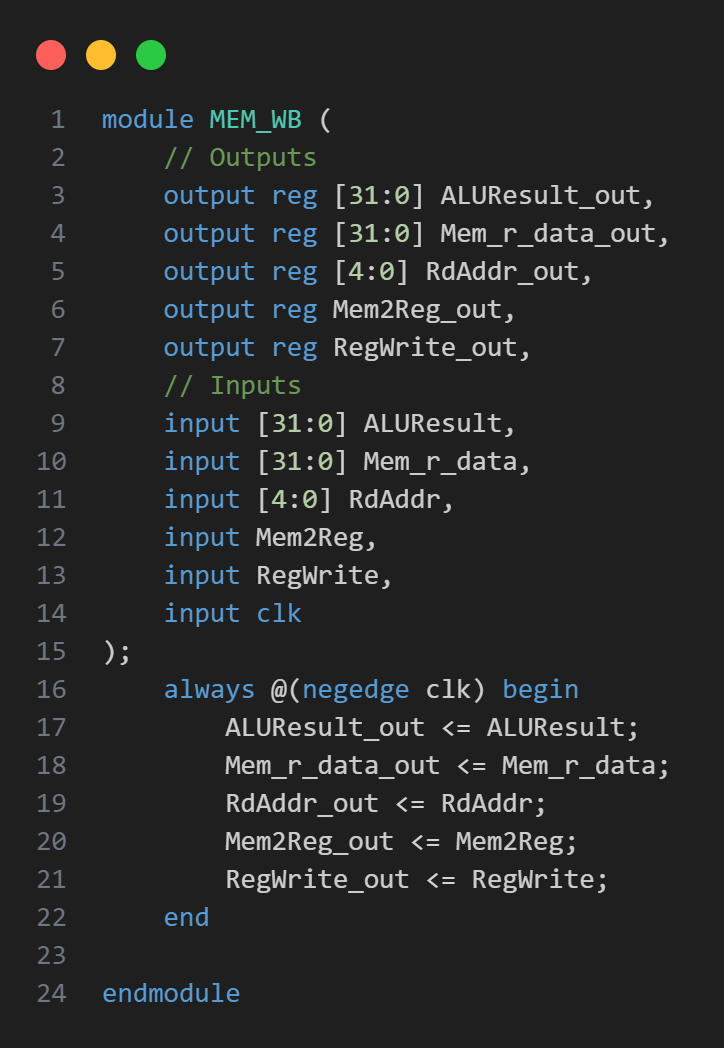
Outputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| Instruction | 32 bits | Output instruction fetched from memory. |

### Explanation

Concatenates four bytes from the memory array (big endian) to form a 32-bit instruction.

## MEM\_WB



### Description

Hold the data and control signals between the Memory (MEM) stage and the Write-Back (WB) stage.

### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| ALUResult | 32 bits | Input result from the ALU. |
| Mem\_r\_data | 32 bits | Input data read from memory. |
| RdAddr | 5 bits | Input address of the destination register. |
| Mem2Reg | 1 bit | Input control signal for memory to register write. |
| RegWrite | 1 bit | Input control signal enabling the register write. |
| clk | 1 bit | Clock signal. |

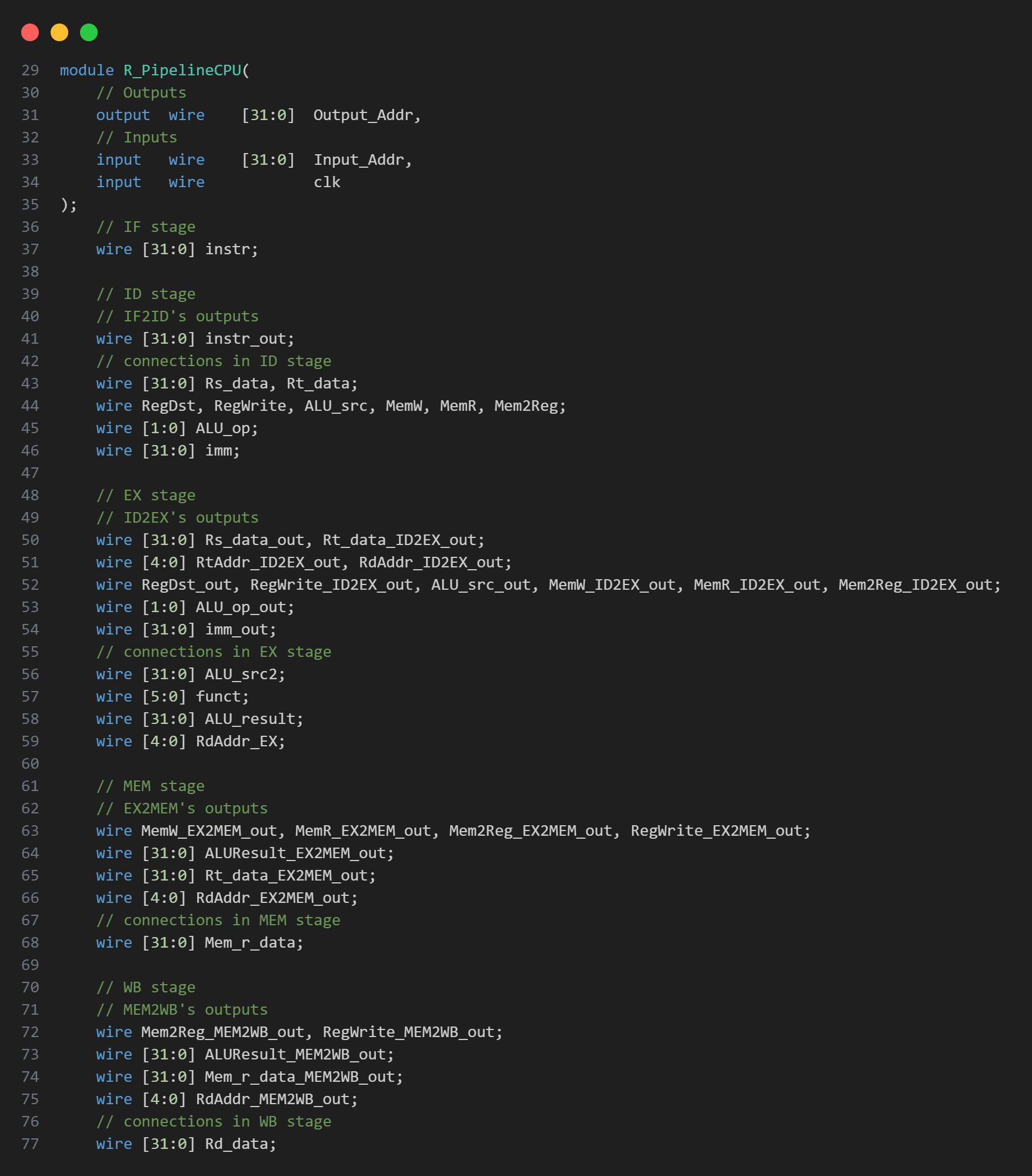
Outputs:

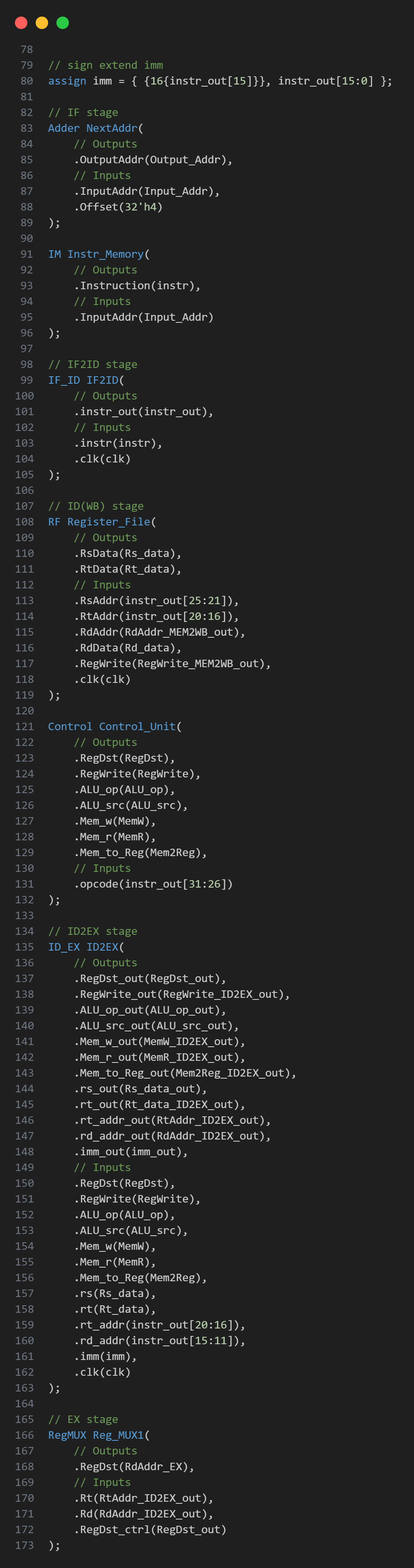
|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| ALUResult\_out | 32 bits | Output result from the ALU to be used in the WB stage. |
| Mem\_r\_data\_out | 32 bits | Output data read from memory to be used in the WB stage. |
| RdAddr\_out | 5 bits | Output address of the destination register. |
| Mem2Reg\_out | 1 bit | Output control signal for memory to register write. |
| RegWrite\_out | 1 bit | Output control signal enabling the register write. |

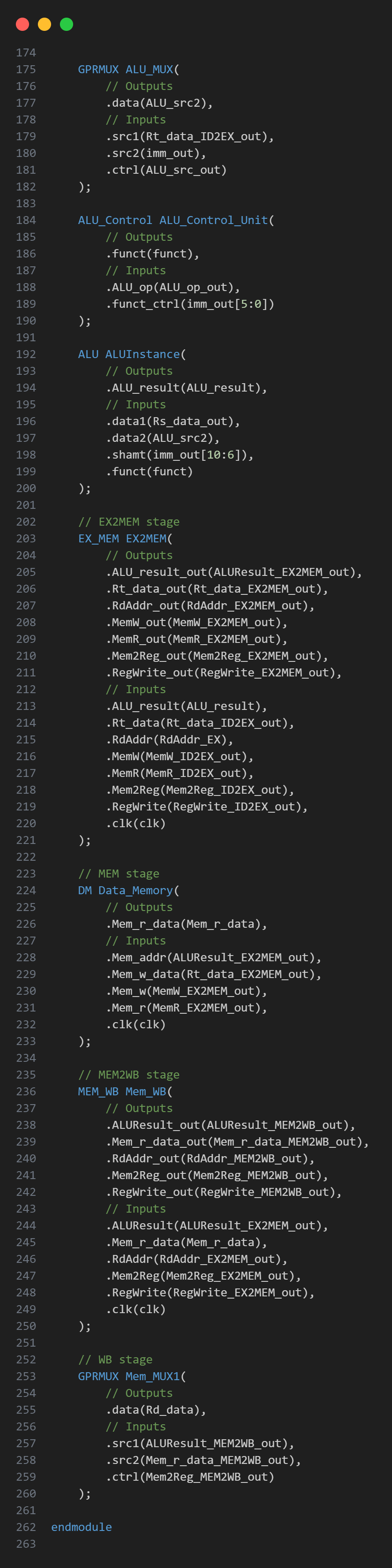
### Explanation

Holds the input values and control signals on the falling edge of the clock (**negedge clk**).

## R\_PipelineCPU







### Description

A pipelined CPU architecture with five pipeline stages: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM), and Write-Back (WB)

### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| Input\_Addr | 32 | PC |
| clk | 1 | Clock signal |

Outputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| Output\_Addr | 32 | PC + 4 |

### Explanation

Line 36~77

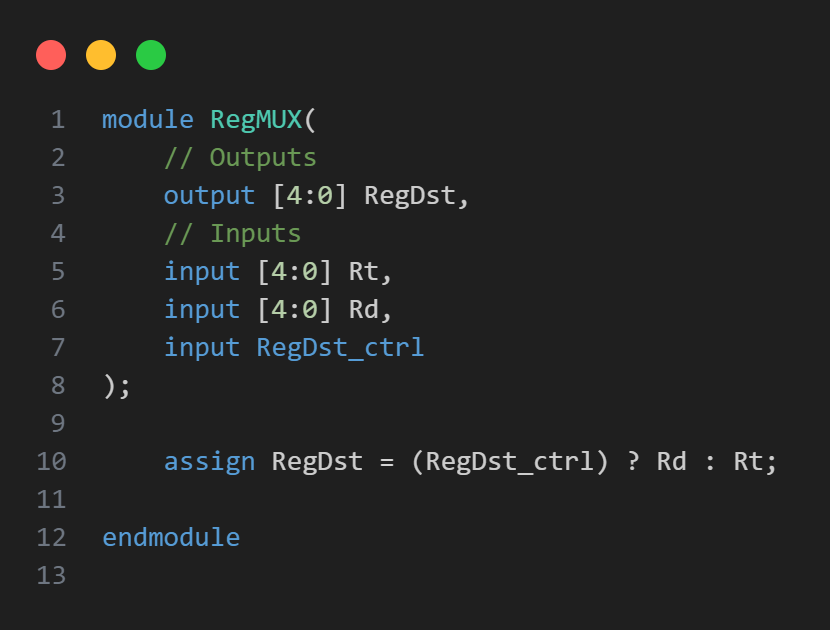
Wires that connecting the components. They are sorted by their corresponding stage.

Line 78~

Components’ instances. They are mostly sorted by their corresponding stage.

Each pipeline stage has its corresponding registers (**IF\_ID**, **ID\_EX**, **EX\_MEM**, **MEM\_WB**) to store intermediate results and control signals between pipeline stages.

## RegMUX



### Description

For selecting the destination register address.

### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| Rt | 5 bits | Input register address representing the source register. |
| Rd | 5 bits | Input register address representing the destination register. |
| RegDst\_ctrl | 1 bit | Input control signal for selecting the destination register. |

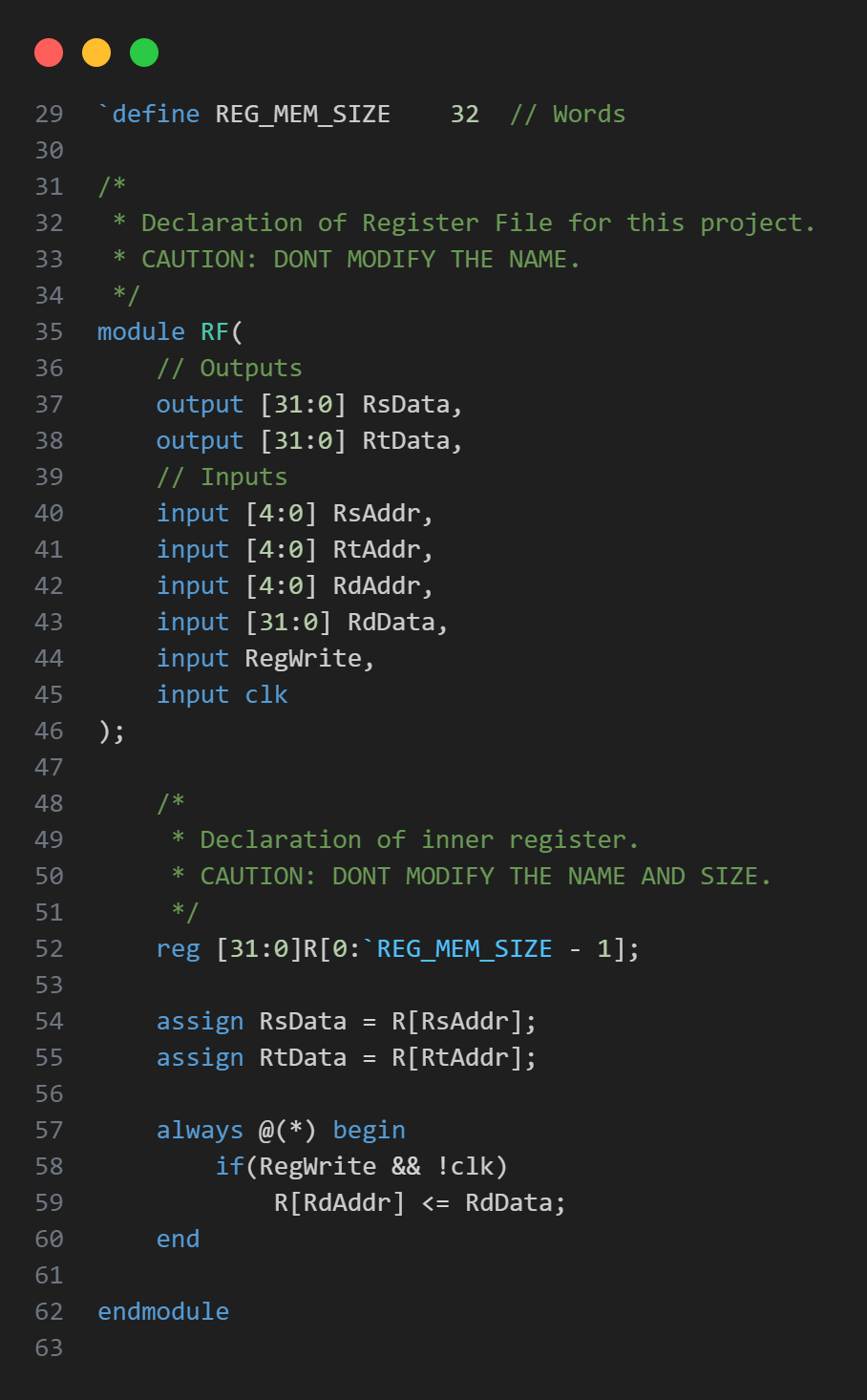
Outputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| RegDst | 5 bits | Output selected register address. |

### Explanation

Using RegDst\_ctrl to selects either the source register address (**Rt**) or the destination register address (**Rd**) as the output.

## RF



### Description

Register file.

### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| RsAddr | 5 bits | Input address for reading data from the register file (**RsData**). |
| RtAddr | 5 bits | Input address for reading data from the register file (**RtData**). |
| RdAddr | 5 bits | Input address for writing data to the register file. |
| RdData | 32 bits | Input data to be written to the register specified by **RdAddr**. |
| RegWrite | 1 bit | Input control signal for enabling register write. |
| clk | 1 bit | Clock signal. |

Outputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| RsData | 32 bits | Output data from the register specified by **RsAddr**. |
| RtData | 32 bits | Output data from the register specified by **RtAddr**. |

### Explanation

* Constantly read data
* Write only when **RegWrite** is asserted and **clk** is deasserted.

# Part 2

## Changes

It’s the same as Part 1. Except “R\_PipelineCPU” is renamed to “I\_PipelineCPU.”

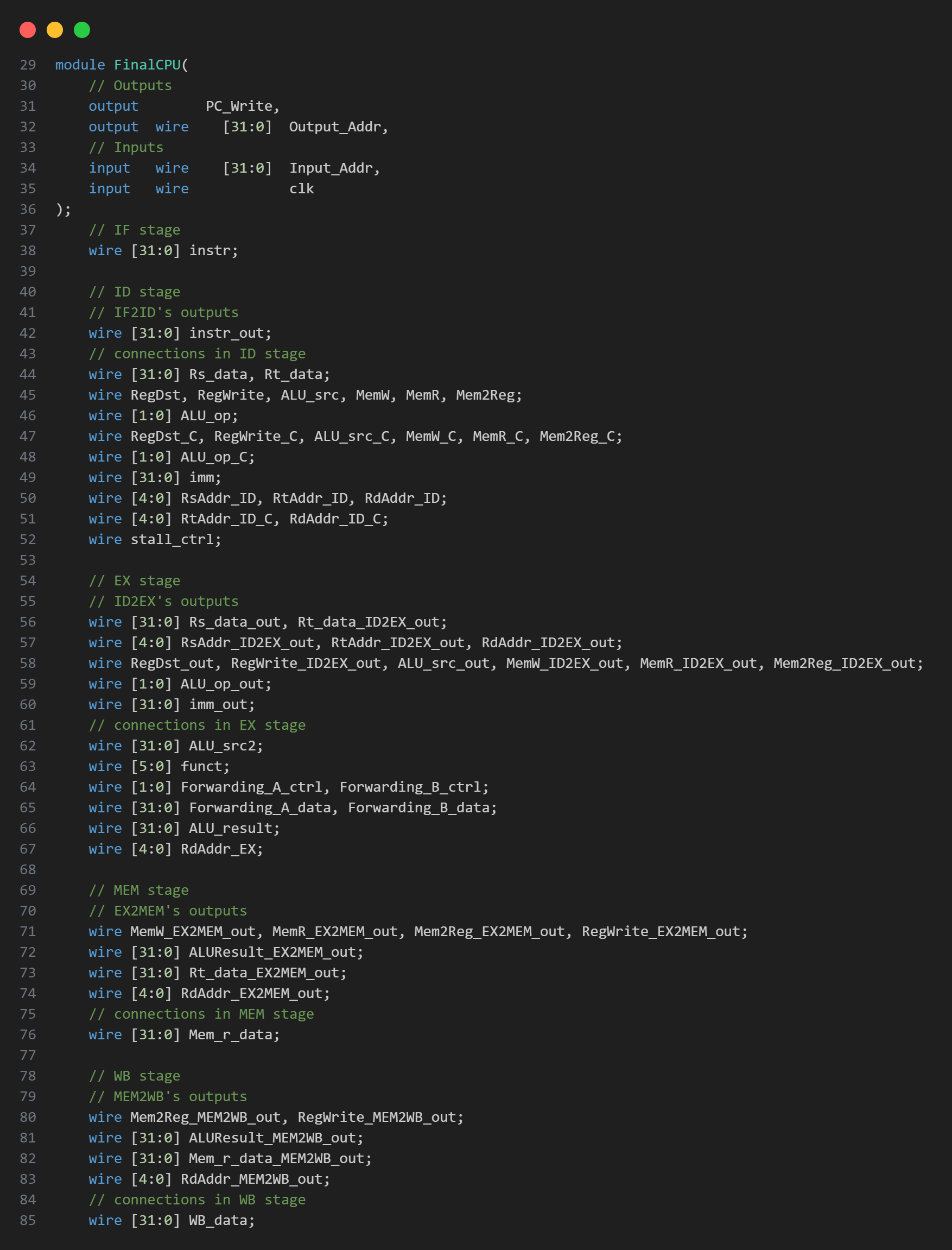
# Part 3

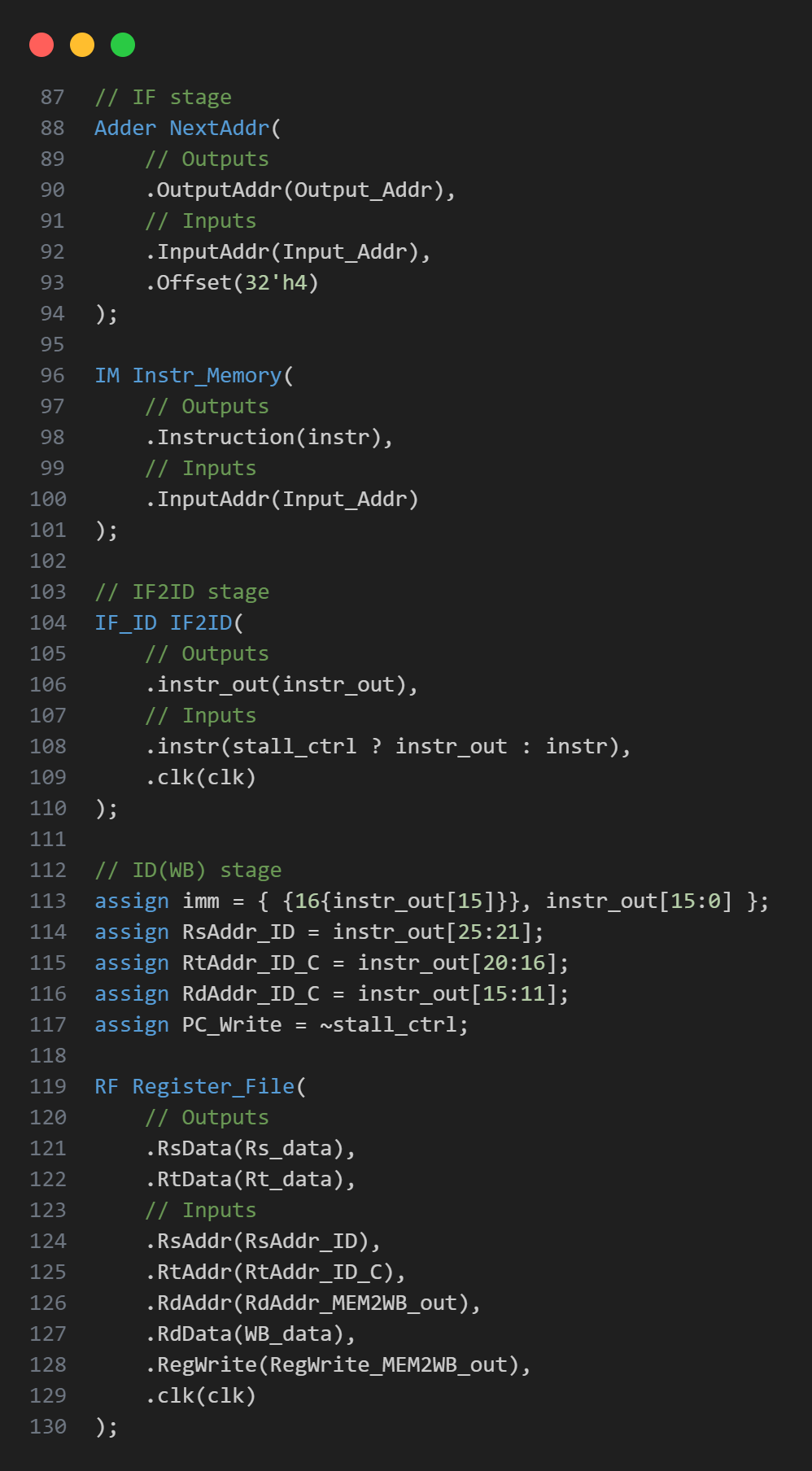
## Changes

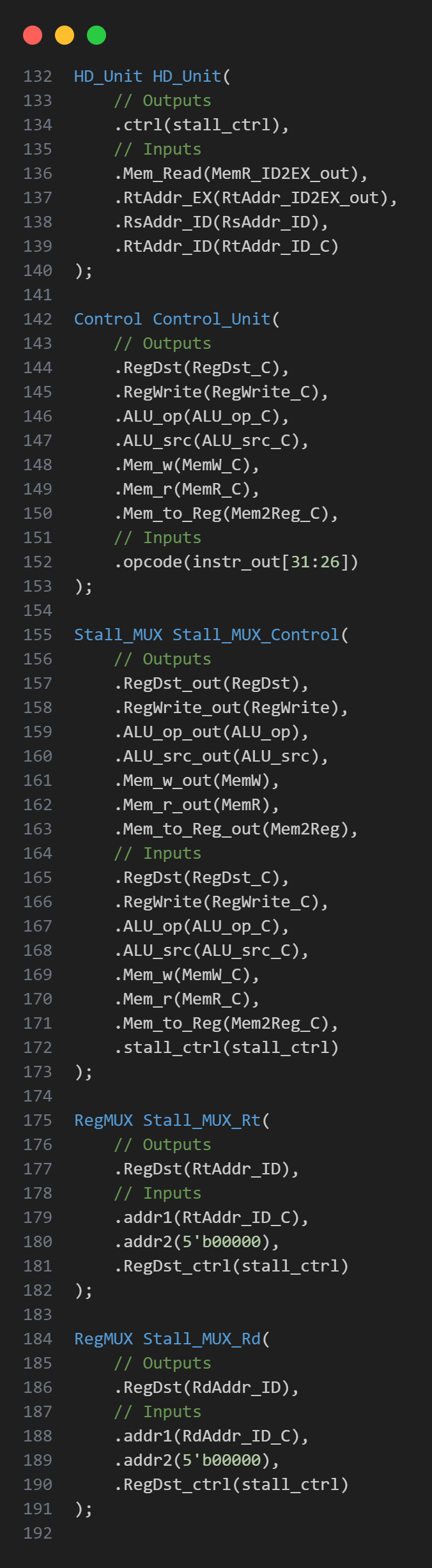
1. Rename “GPRMUX” to “DMUX” which stands for double sources mux.
2. Rename RegMUX’s input addresses as addr1 and addr2 respectively for more accurate discription.
3. “ID\_EX” add **rs\_addr** as input and **rs\_addr\_out** as output for detecting data hazard.

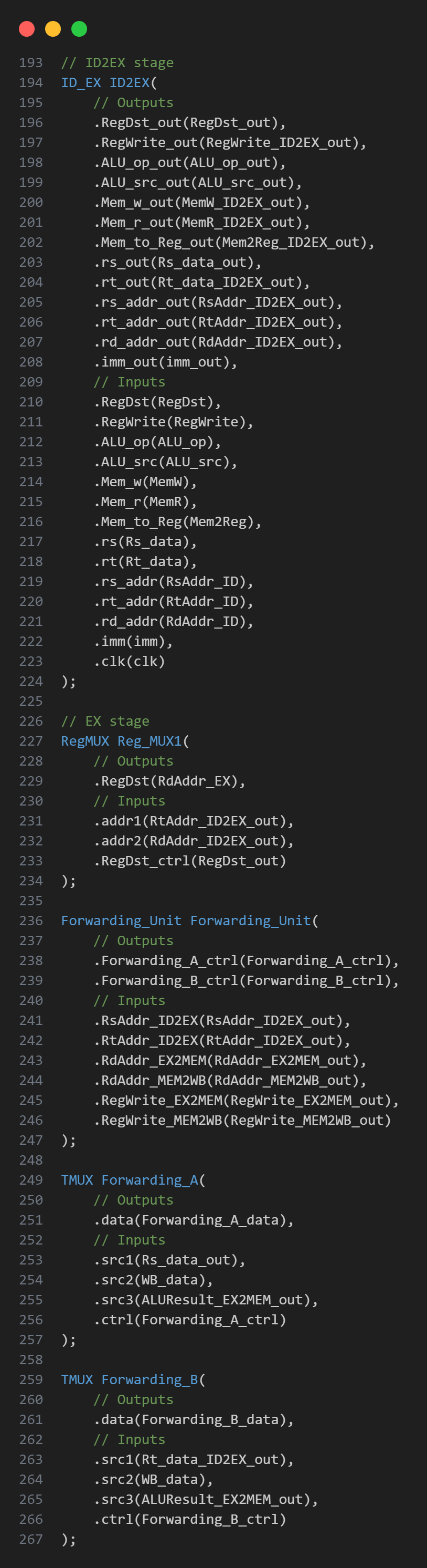
## New modules

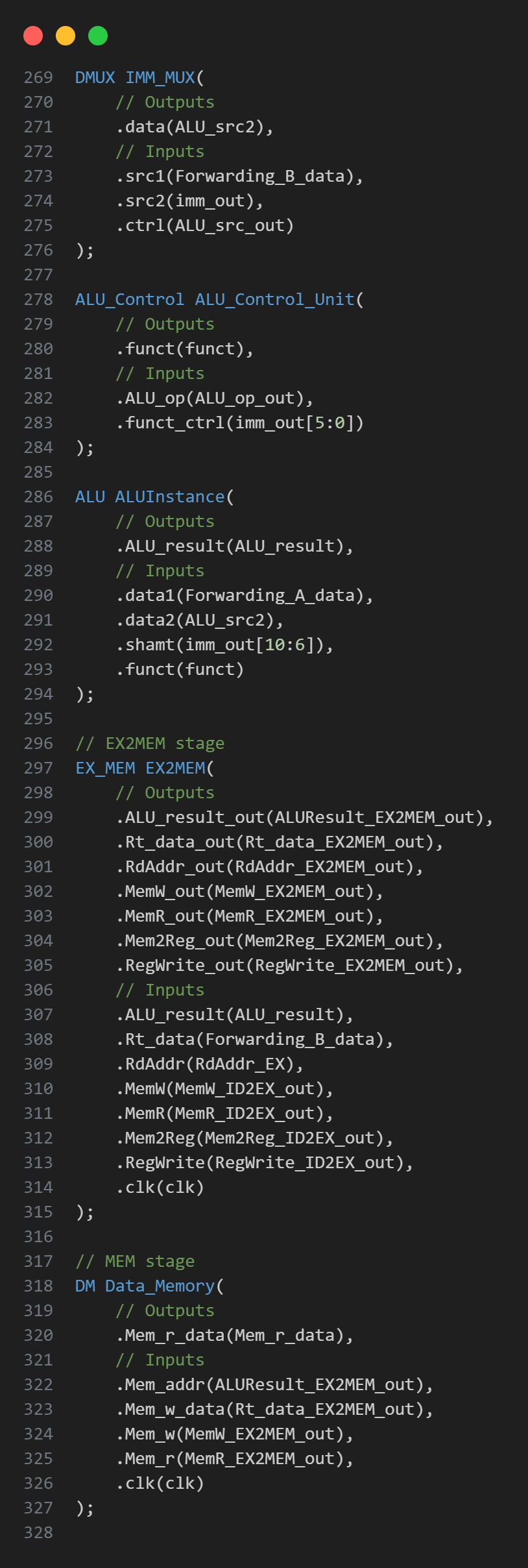
### FinalCPU

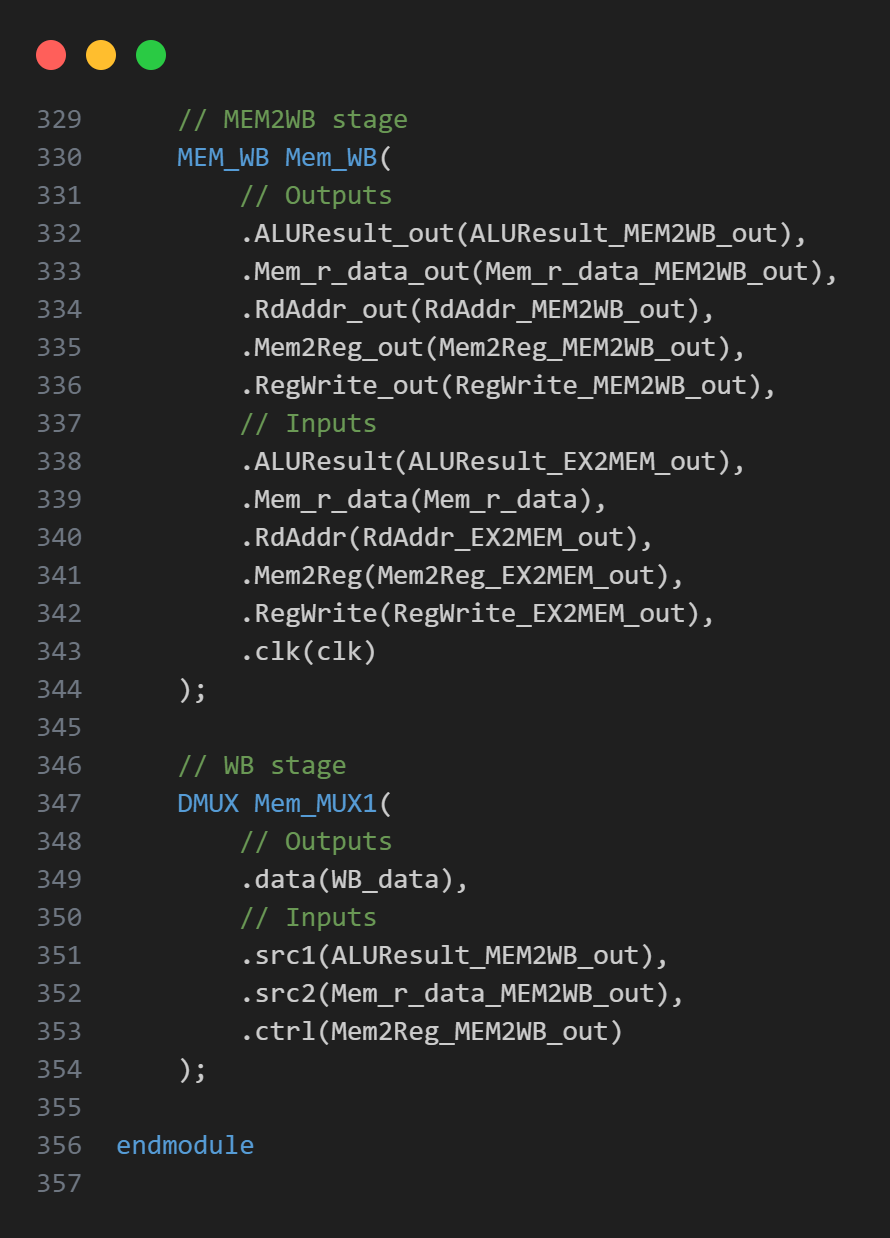












#### Description

A pipelined CPU architecture with five pipeline stages: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM), and Write-Back (WB)

#### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| Input\_Addr | 32 | PC |
| clk | 1 | Clock signal |

Outputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| PC\_Write | 1 | ~stall |
| Output\_Addr | 32 | PC + 4 |

#### Explanation

Line 37~85

Wires that connecting the components. They are sorted by their corresponding stage.

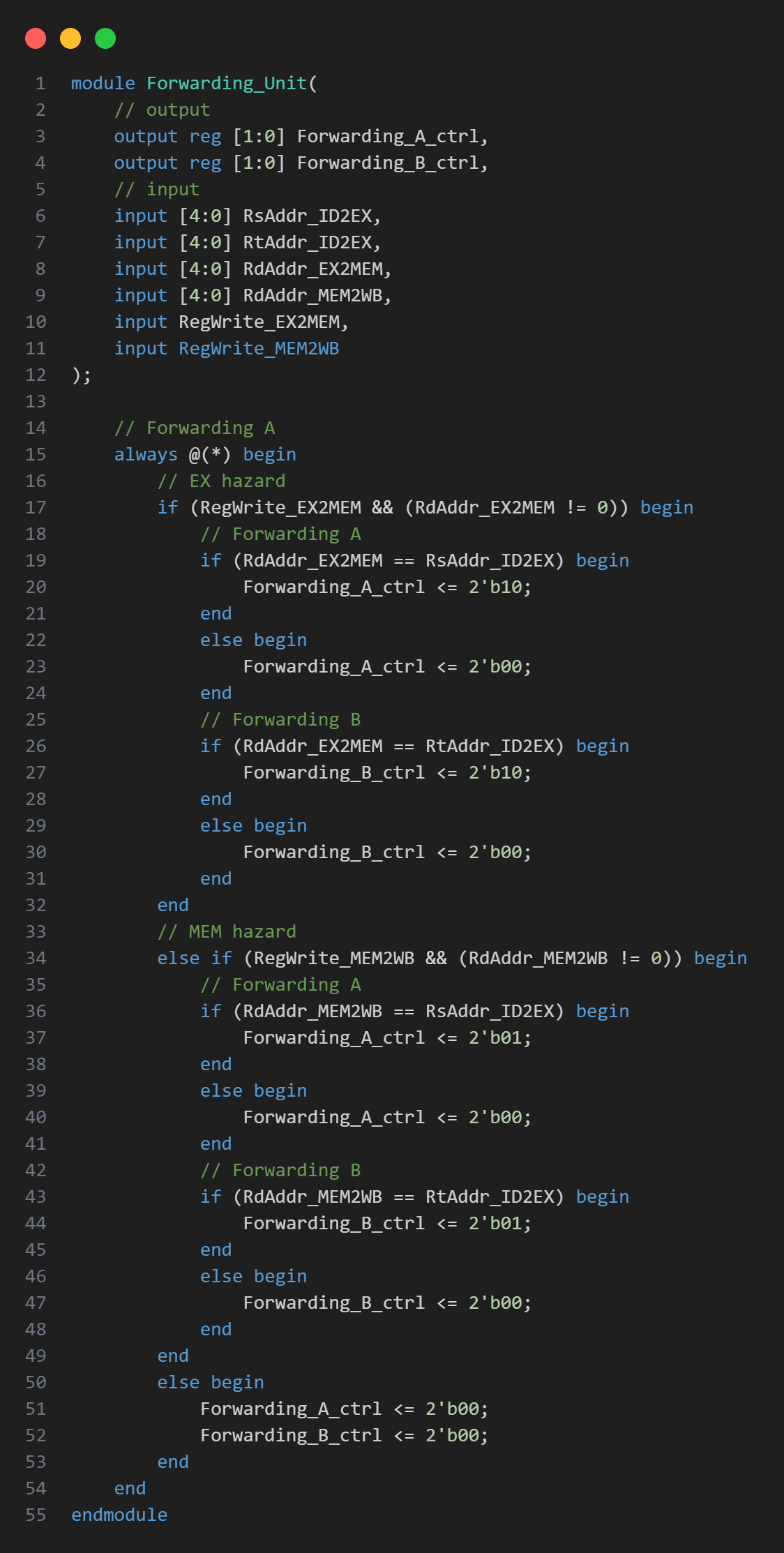
Line 86~

Components’ instances. They are mostly sorted by their corresponding stage.

Each pipeline stage has its corresponding registers (**IF\_ID**, **ID\_EX**, **EX\_MEM**, **MEM\_WB**) to store intermediate results and control signals between pipeline stages.

Comparing to previous CPUs, this one uses data forwarding and stall to avoid data hazard.

### Forwarding\_Unit



#### Description

generates control signals (**Forwarding\_A\_ctrl** and **Forwarding\_B\_ctrl**) to indicate whether to forward data from the execution stage or memory stage to the instruction EX stage.

#### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| RsAddr\_ID2EX | 5 bits | Input address of source register (**Rs**) in the instruction decode/execute stage. |
| RtAddr\_ID2EX | 5 bits | Input address of target register (**Rt**) in the instruction decode/execute stage. |
| RdAddr\_EX2MEM | 5 bits | Input address of destination register (**Rd**) in the execution stage. |
| RdAddr\_MEM2WB | 5 bits | Input address of destination register (**Rd**) in the memory stage. |
| RegWrite\_EX2MEM | 1 bit | Input control signal indicating register write in the MEM stage. |
| RegWrite\_MEM2WB | 1 bit | Input control signal indicating register write in the WB stage. |

Outputs:

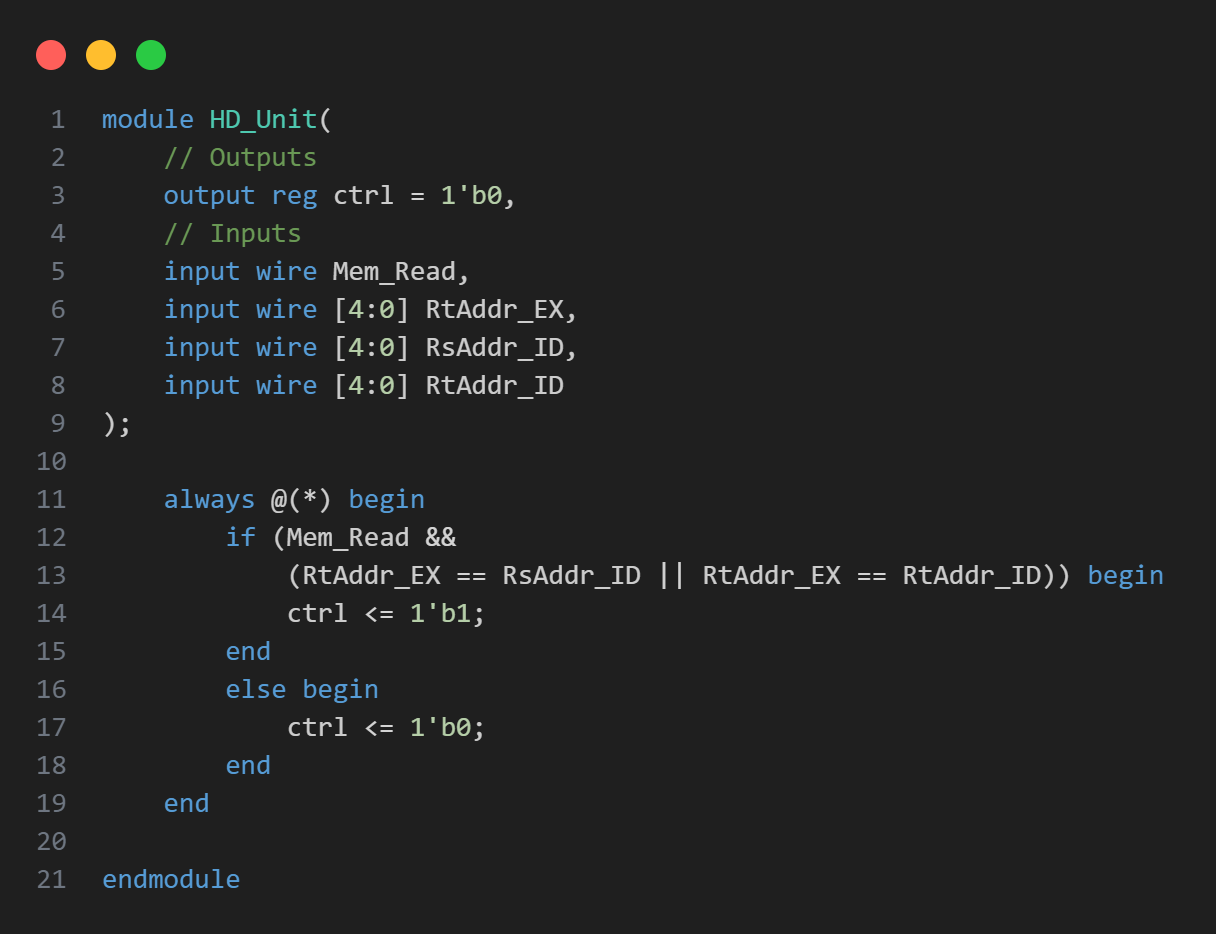
|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| Forwarding\_A\_ctrl | 2 bits | Output control signal for forwarding data to operand A in the instruction EX stage. |
| Forwarding\_B\_ctrl | 2 bits | Output control signal for forwarding data to operand B in the instruction EX stage. |

#### Explanation

Detects data hazards by comparing the destination register addresses.

* ID/EX: no forwarding, just from the register file (00)
* EX/MEM: forwarded data from the prior ALU results (10)
* MEM/WB: from data memory or an earlier ALU result (01)

### HD\_Unit



#### Description

Hazard Detection Unit identifies the hazard that cannot be solved by data forwarding.

#### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| Mem\_Read | 1 bit | Input signal indicating a memory read operation in the EX stage. |
| RtAddr\_EX | 5 bits | Input address of the target register (**Rt**) in the execution stage (**EX**). |
| RsAddr\_ID | 5 bits | Input address of the source register (**Rs**) in the instruction decode stage (**ID**). |
| RtAddr\_ID | 5 bits | Input address of the target register (**Rt**) in the instruction decode stage (**ID**). |

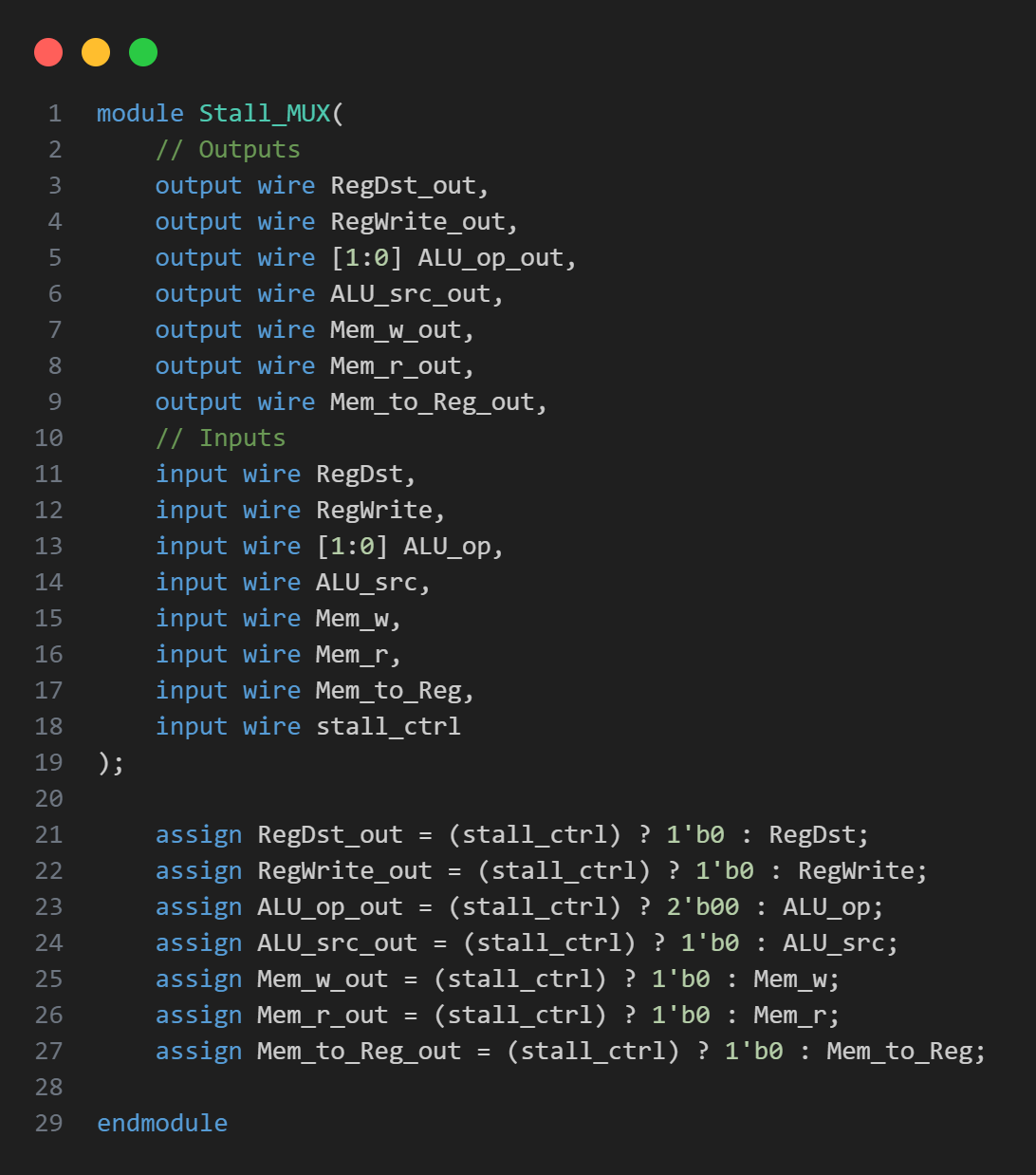
Outputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| ctrl | 1 bit | Output control signal indicating the presence of a hazard. |

#### Explanation

Sets the control signal (**ctrl**) to indicate the presence of a hazard, enabling stall operation.

### Stall\_MUX



#### Description

Control multiplexer outputs based on a stall control signal (**stall\_ctrl**)

#### IO

Inputs:

All outputs of Control

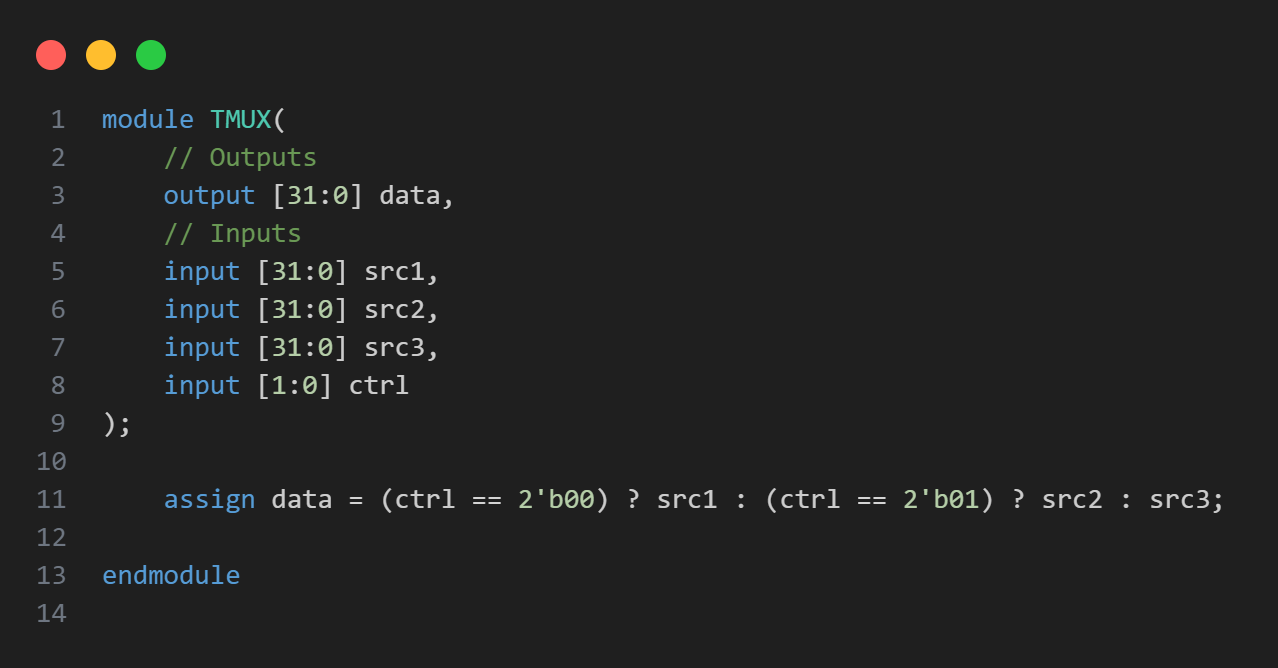
Outputs:

Control signals that have the same amount to Control’s output.

#### Explanation

Utilizes **assign** statements to control the outputs of multiplexers based on the stall control signal (**stall\_ctrl**). When a stall is required, all outputs are forced to a default state.

### TMUX



#### Description

Triple MUX. Selecting among 3 sources.

#### IO

Inputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| src1 | 32 bits | Input data source 1. |
| src2 | 32 bits | Input data source 2. |
| src3 | 32 bits | Input data source 3. |
| ctrl | 2 bits | Control signal determining the selected input data source. |

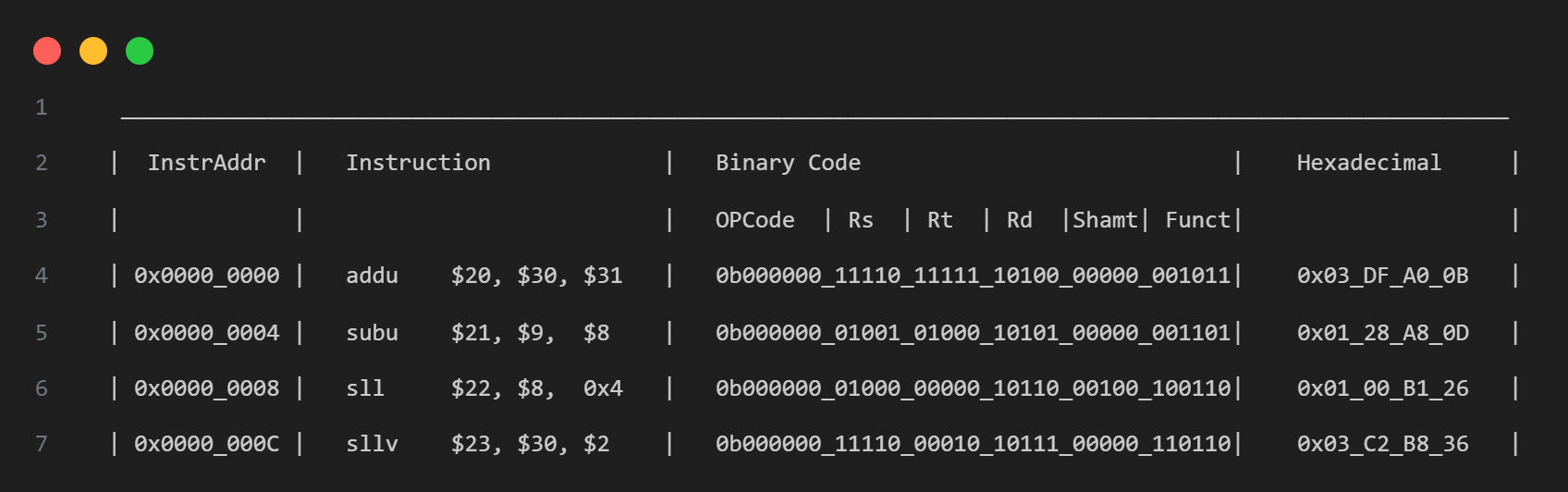
Outputs:

|  |  |  |
| --- | --- | --- |
| Port | Width | Description |
| data | 32 bits | Output data resulting from the multiplexing operation. |

#### Explanation

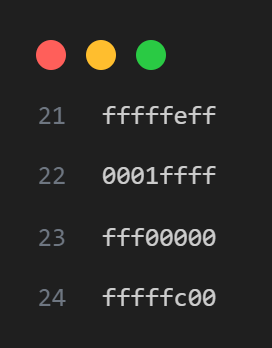
* ctrl == 0, select src1
* ctrl == 1, select src2
* ctrl == 2, select src3

# Part 1 Test Program

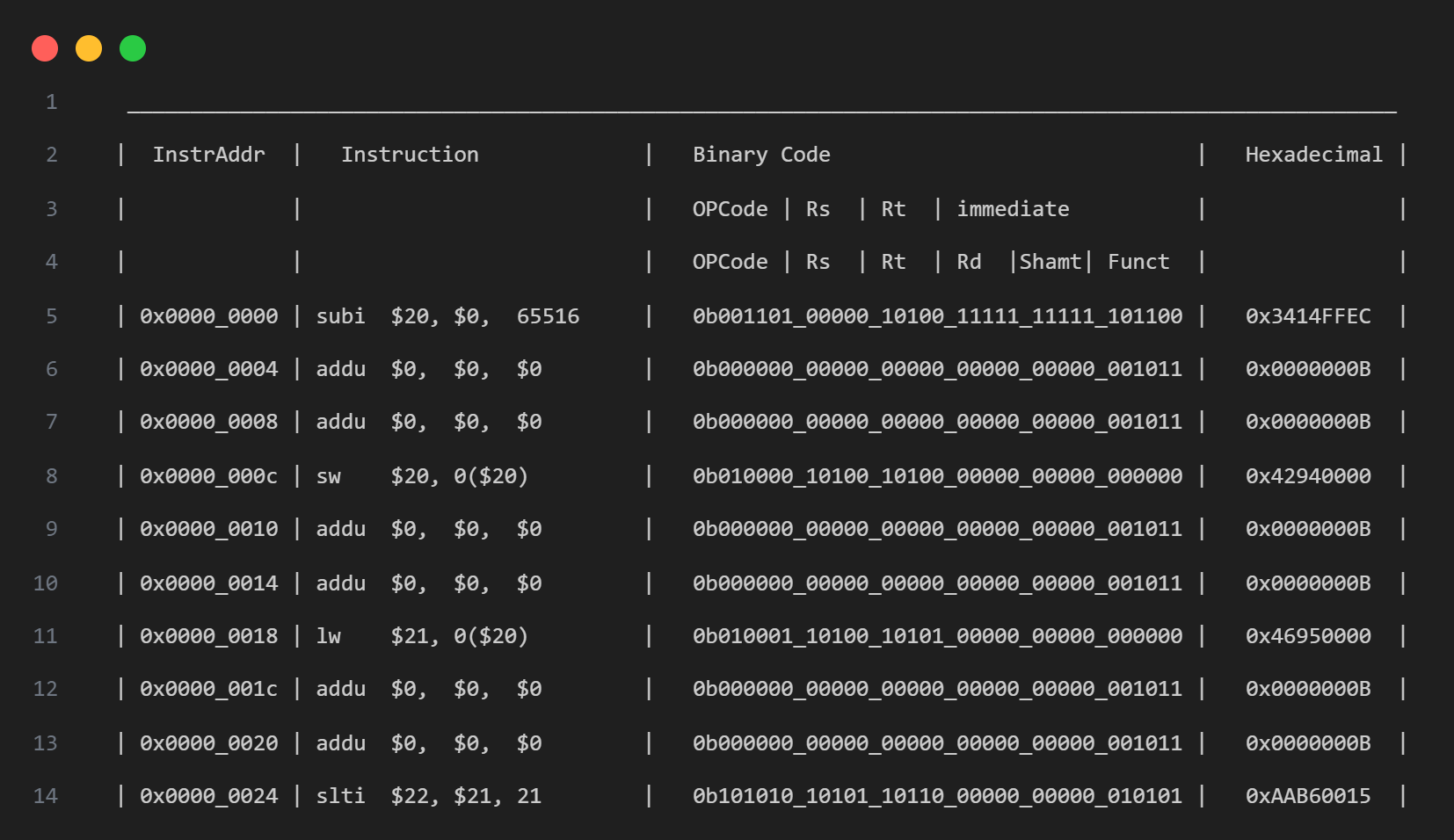


1. R[20]=R[30]+R[31]=FFFF\_FF00+ FFFF\_FFFF=FFFF\_FEFF
2. R[21]=R[9]-R[8]=0000\_FFFF- FFFF\_0000=0001\_FFFF
3. R[22]=R[8]<<shamt=FFFF\_0000<<4=FFF0\_0000
4. R[23]=R[30]<<R[2][4:0]=FFFF\_FF00<<02=FFFF\_FC00

## RF

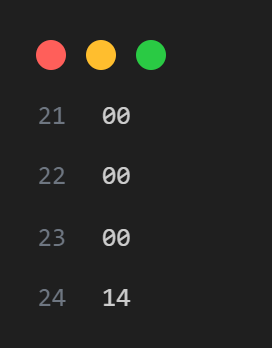


# Part 2 Test Program

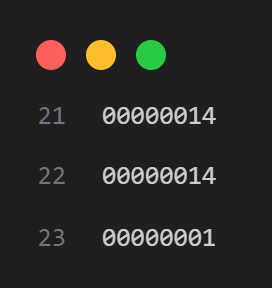


1. R[20]=R[0]-65516=0+0x14=0x14
2. R[0]=R[0]+R[0]=0+0=0, dummy op for avoiding data hazard.
3. R[0]=R[0]+R[0]=0+0=0
4. Mem[0+R[20]]=Mem[0x14]=$20=0x14
5. R[0]=R[0]+R[0]=0+0=0
6. R[0]=R[0]+R[0]=0+0=0
7. R[21]=Mem[0+R[20]]=Mem[0x14]=0x14
8. R[0]=R[0]+R[0]=0+0=0
9. R[0]=R[0]+R[0]=0+0=0
10. R[22]=(R[21]<21)?1:0=1

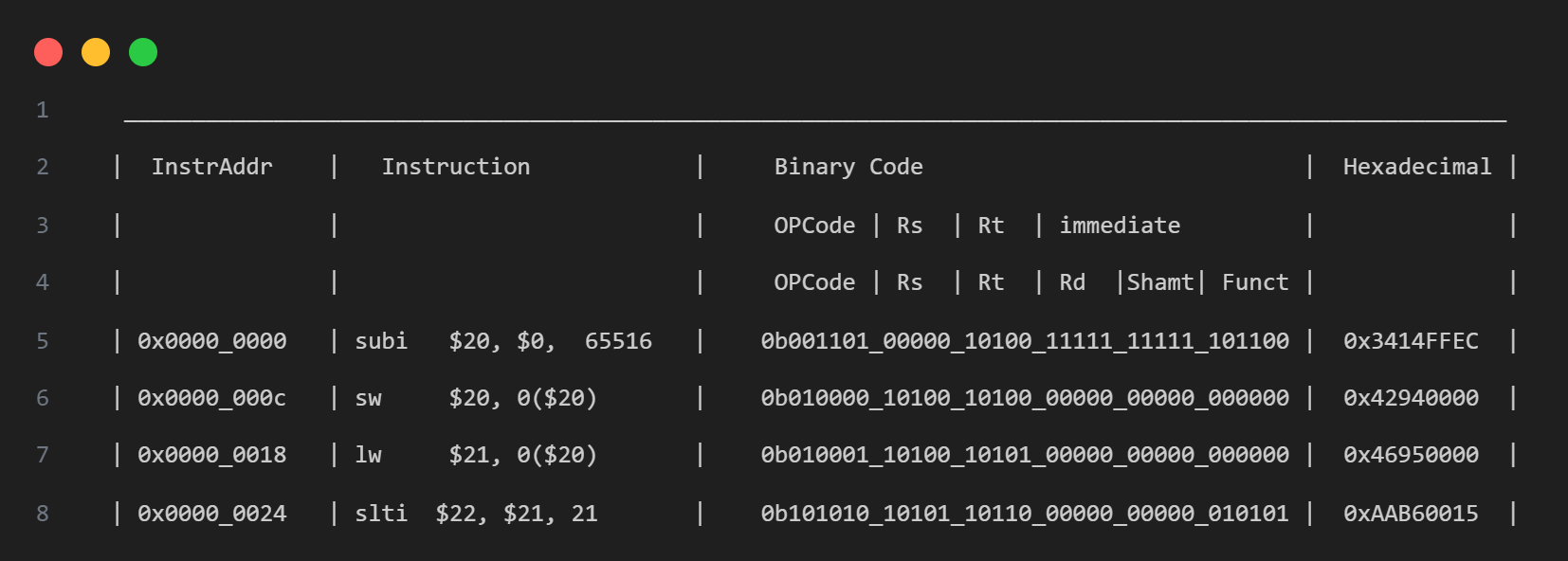
## DM



## RF

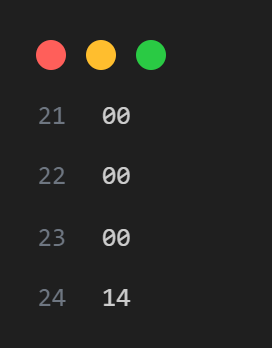


# Part 3 Test Program

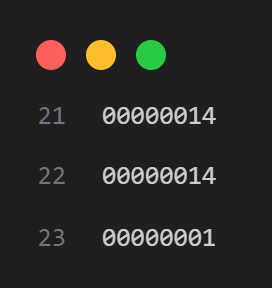


1. R[20]=R[0]-65516=0+0x14=0x14
2. Mem[0+R[20]]=Mem[0x14]=$20=0x14
3. R[21]=Mem[0+R[20]]=Mem[0x14]=0x14
4. R[22]=(R[21]<21)?1:0=1

## DM



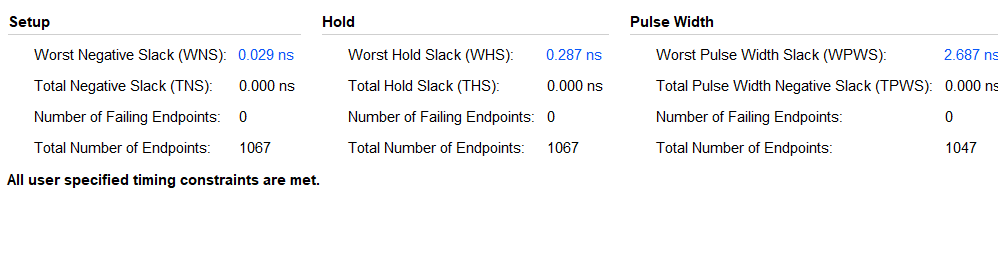
## RF

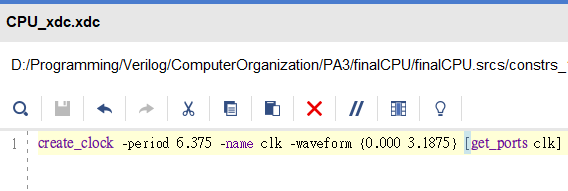


# Simulation Result

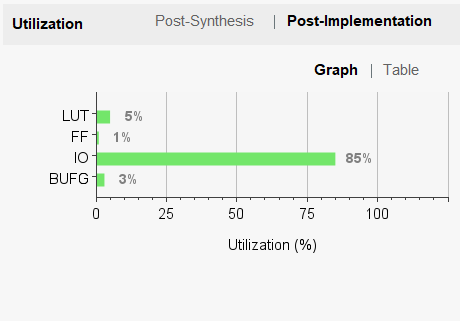
## Timing

Clock period = 6.375 ns





## Utilization



## Power

