Computer Organization

Single-cycle CPU

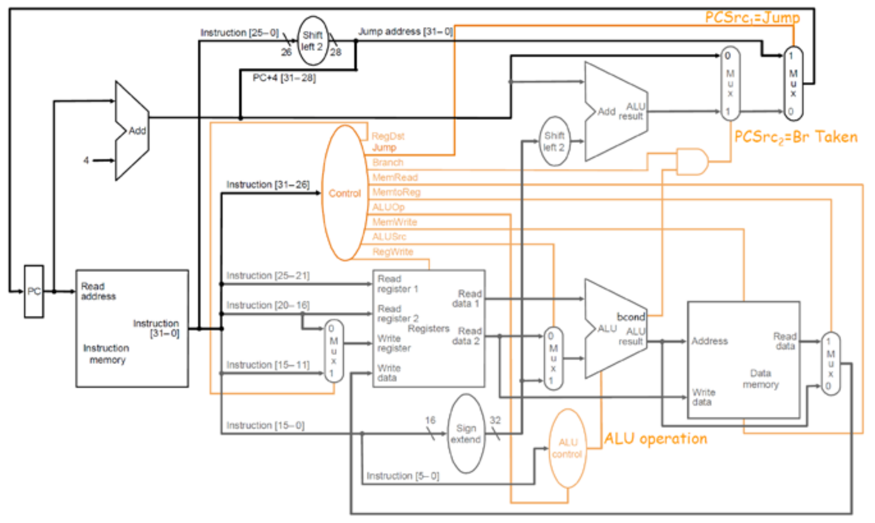
Eunjin Baek, Pyeongsu Park
High Performance Computer System (HPCS) Lab
April 3rd, 2018

Goals

- Understand the basic CPU structure.
- Understand the roles of **the datapath and the control unit**.
- Implement a single-cycle CPU in Verilog.

SNU EE430.322
Spring '18 L5-28
Jangwoo Kim

Putting It All together



**Based on figures from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

JAL, JR, JALR omitted

← Implement this slide!

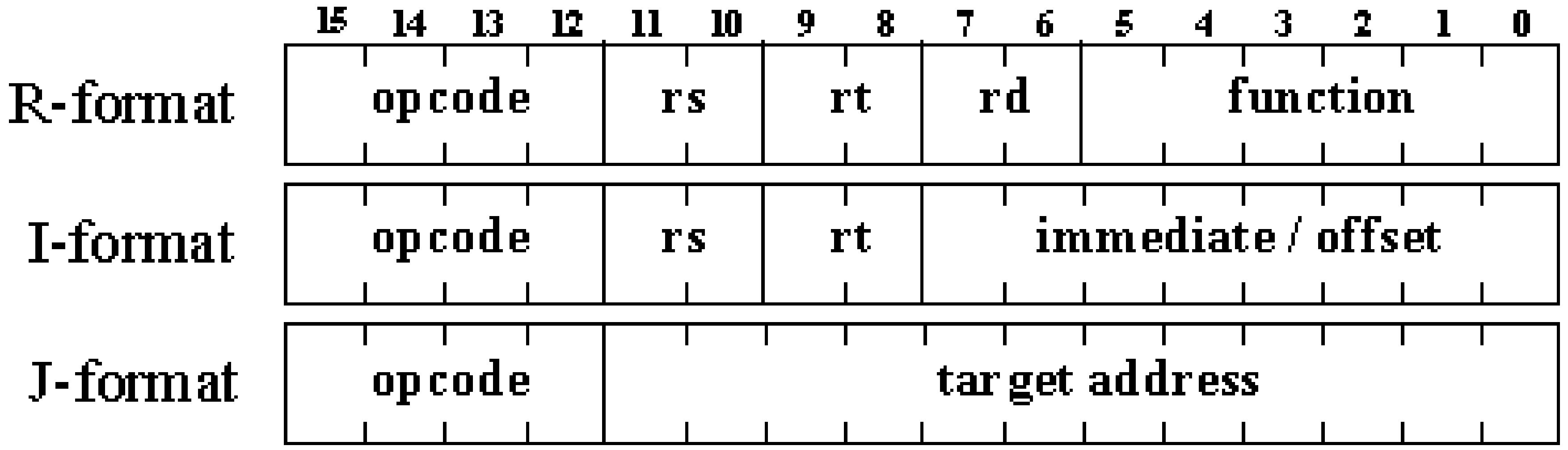
CPU structure: ISA

- Instruction Set Architecture (ISA)
  - Programmer visible state
  - User manual of the computer

- We will use **TSC ISA** instead of complicated MIPS ISA
  - 16-bit RISC ISA
  - Similar to MIPS, but more simplified
  - Refer to the manual on the eTL board

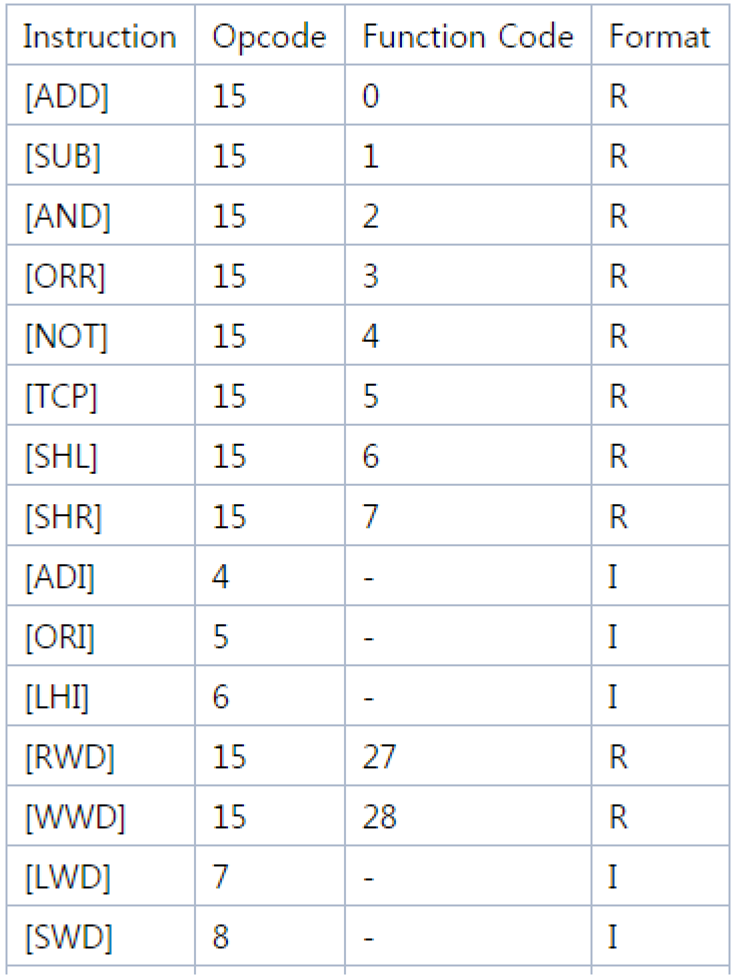
TSC ISA

- Instruction types
  - R, I, and J-format
  - Similar to MIPS



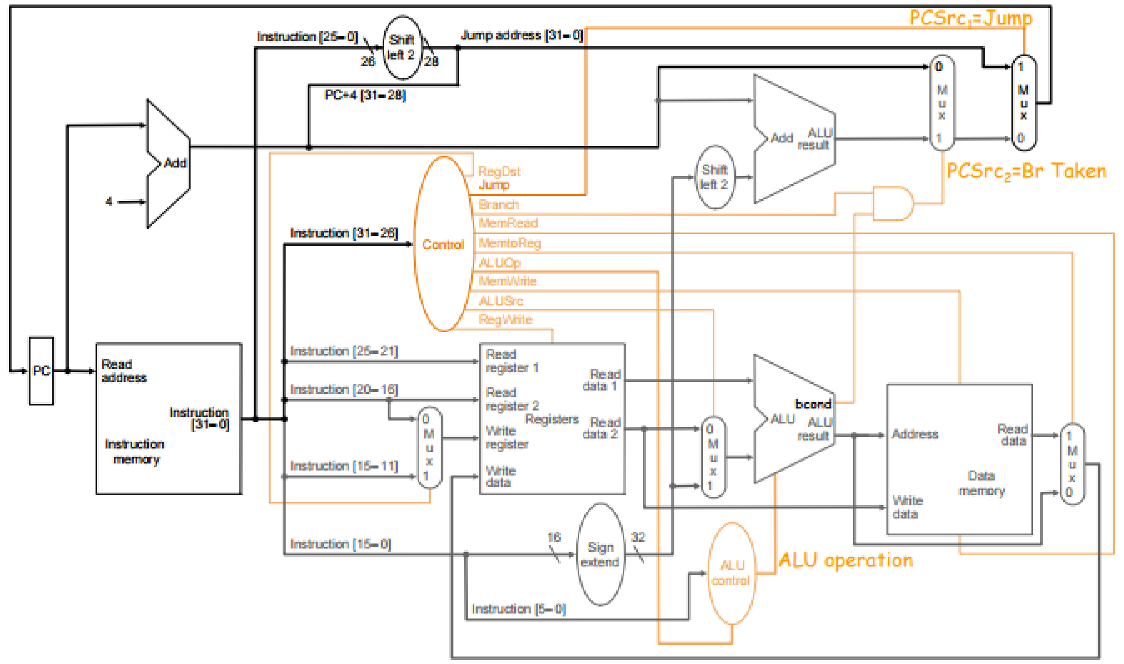
CPU structure: TSC ISA

- Instruction opcodes
  - Refer to the manual!

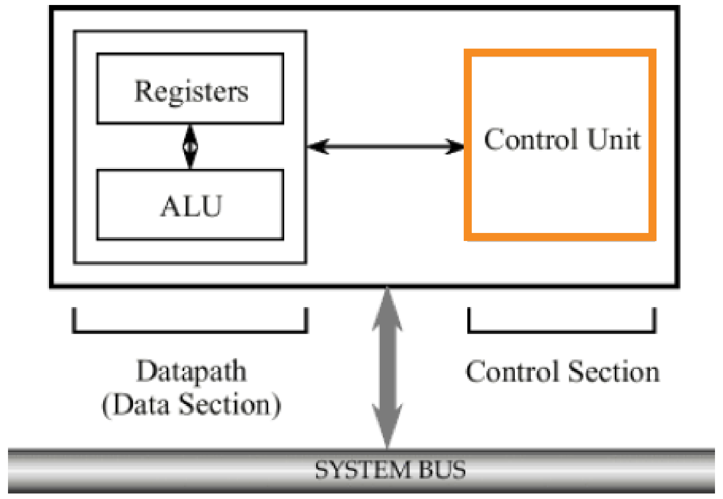


Instruction	Opcode	Function Code	Format
[ADD]	15	0	R
[SUB]	15	1	R
[AND]	15	2	R
[ORR]	15	3	R
[NOT]	15	4	R
[TCP]	15	5	R
[SHL]	15	6	R
[SHR]	15	7	R
[ADI]	4	-	I
[ORI]	5	-	I
[LHI]	6	-	I
[RWD]	15	27	R
[WWD]	15	28	R
[LWD]	7	-	I
[SWD]	8	-	I

How to Make CPU?



Simplified



- Generally, CPU consists of two major components
  - **Datapath** (black boxes/lines) – Units in the path of data
  - **Control Unit** (orange boxes/lines) – Units which direct operations

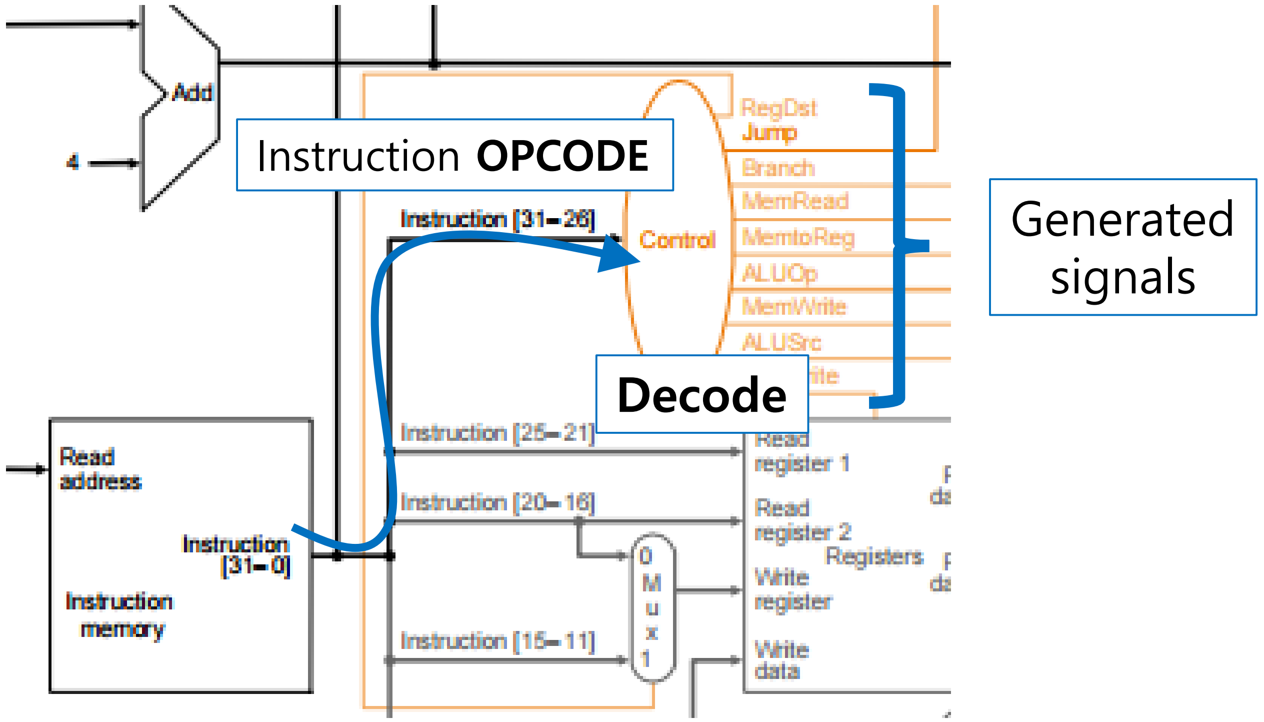
Datapath

- Datapath
  - Calculation: ALU
  - Data management: Register, Memory

❖Register file (RF)
  - Read should be asynchronous to the clock. If you implement the RF synchronously, please change your design for the following projects.

- Chapter 4 of the textbook will be helpful to understand it ☺

Control



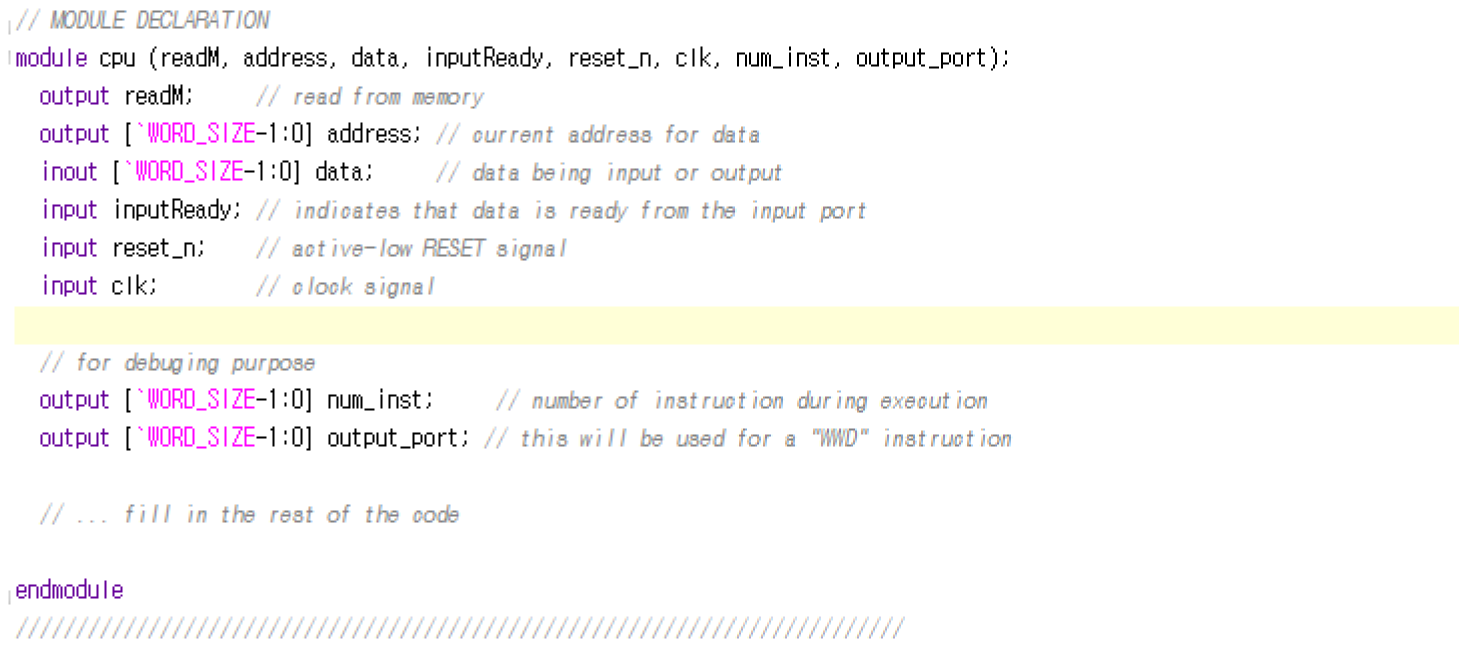
- Control unit
  - Decodes instruction
  - Generate the control signals for the datapath

Assignment

- Implement a *single-cycle* TSC CPU.
  - A CPU that executes **one instruction per cycle**
  - **A datapath** for 16-bit CPU with four registers
  - **A control unit** to generate control signals used by the datapath
  - At this time, it has only five instructions **(ADD, ADI, LHI, JMP, WWD)**

- The datapath & the control unit should be clearly **separated**
  - Build them as two separate modules and connect them in the top module!
  - Control signals should transfer information between the datapath and the control unit.

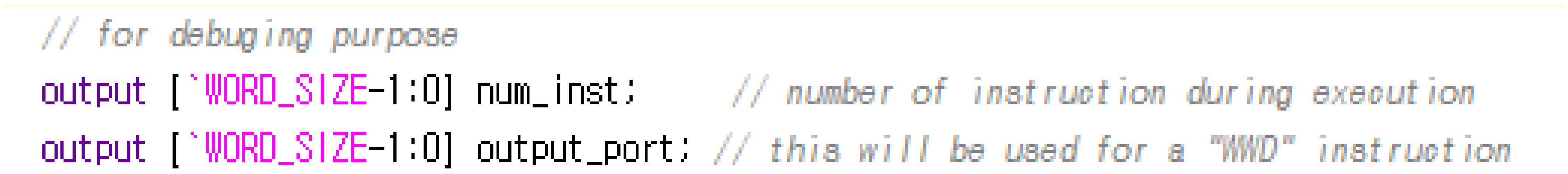
Tips: Implementation



```
// MODULE DECLARATION
module cpu (readM, address, data, inputReady, reset_n, clk, num_inst, output_port);
  output readM;     // read from memory
  output [`WORD_SIZE-1:0] address; // current address for data
  inout [`WORD_SIZE-1:0] data;     // data being input or output
  input inputReady; // indicates that data is ready from the input port
  input reset_n;    // active-low RESET signal
  input clk;        // clock signal

  // for debuging purpose
  output [`WORD_SIZE-1:0] num_inst;     // number of instruction during execution
  output [`WORD_SIZE-1:0] output_port; // this will be used for a "WWD" instruction

  // ... fill in the rest of the code

endmodule
//////////////////////////////////////////////////////////////////////////////
```

- We provide CPU skeleton code
  - Please refer to the comments for usages

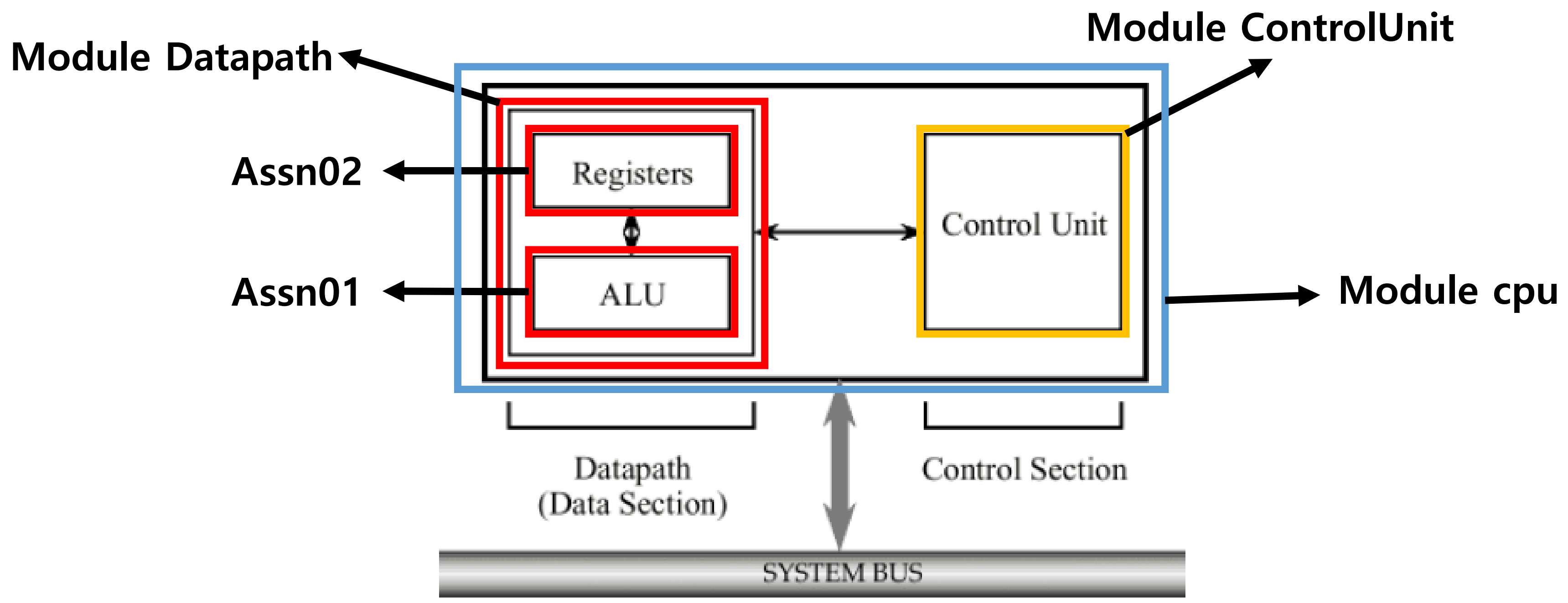
Tips: Implementation



```
// for debuging purpose
output [`WORD_SIZE-1:0] num_inst;      // number of instruction during execution
output [`WORD_SIZE-1:0] output_port; // this will be used for a "WWD" instruction
```

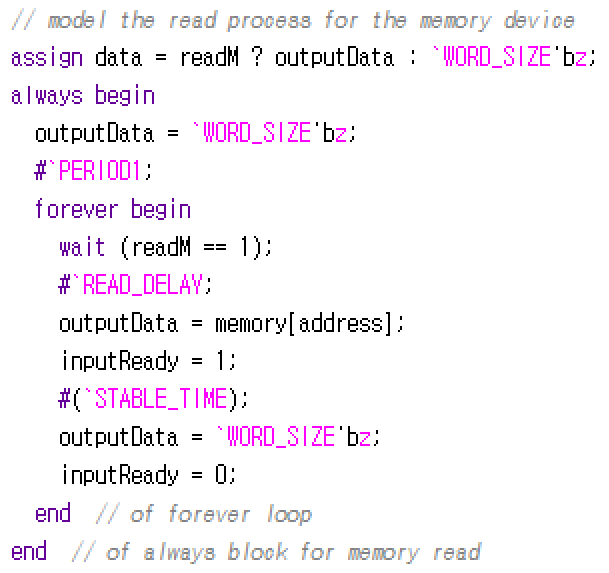
- output **num_inst**
  - The number of instructions during an execution
  - For this project, you need to increment it for every posedge clock signals
- output **output_port**
  - Used only for WWD instructions
  - WWD instructions [WWD $rs] operates output port ← $rs
- Testbench checks num_inst and output_port
  - Please refer to the given testbench file

Tips: Implementation



- Make your own "Datapath & ControlUnit" modules
  - The datapath module should contain an ALU module (Project 1) and a register file (Project 2)
  - The CPU module includes the datapath & the control unit modules

Tips: Testbench

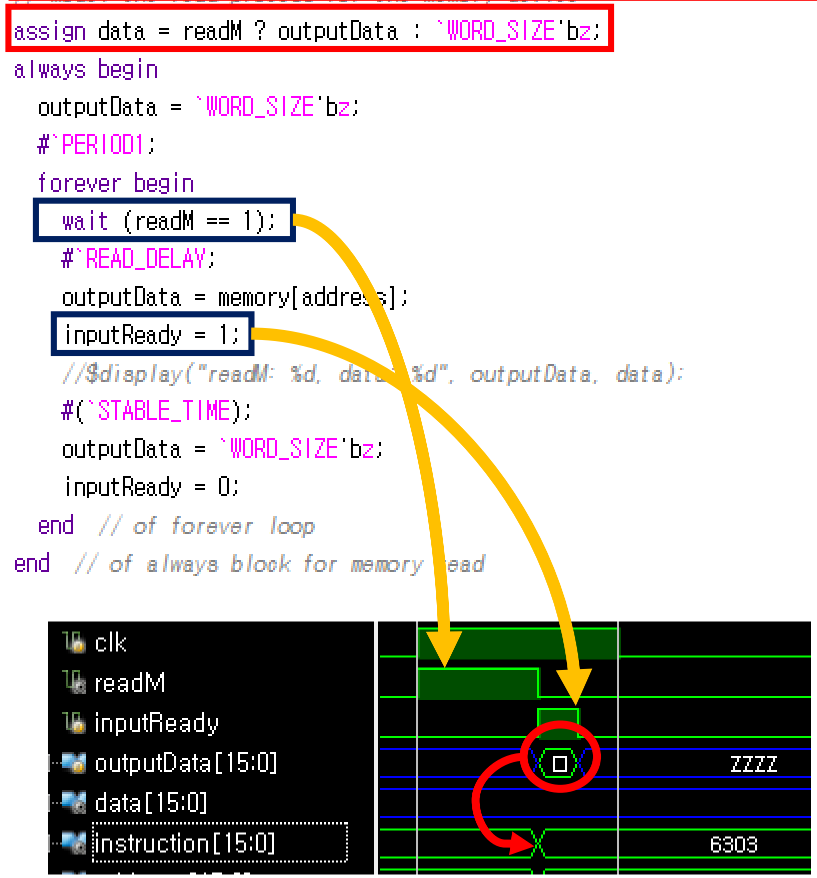


```
// model the read process for the memory device
assign data = readM ? outputData : `WORD_SIZE'bz;
always begin
  outputData = `WORD_SIZE'bz;
  #`PERIOD1;
  forever begin
    wait (readM == 1);
    #`READ_DELAY;
    outputData = memory[address];
    inputReady = 1;
    #(`STABLE_TIME);
    outputData = `WORD_SIZE'bz;
    inputReady = 0;
  end  // of forever loop
end  // of always block for memory read
```

- We also provide a testbench for the CPU module
  - It contains memory code and in-memory instructions for test-cases

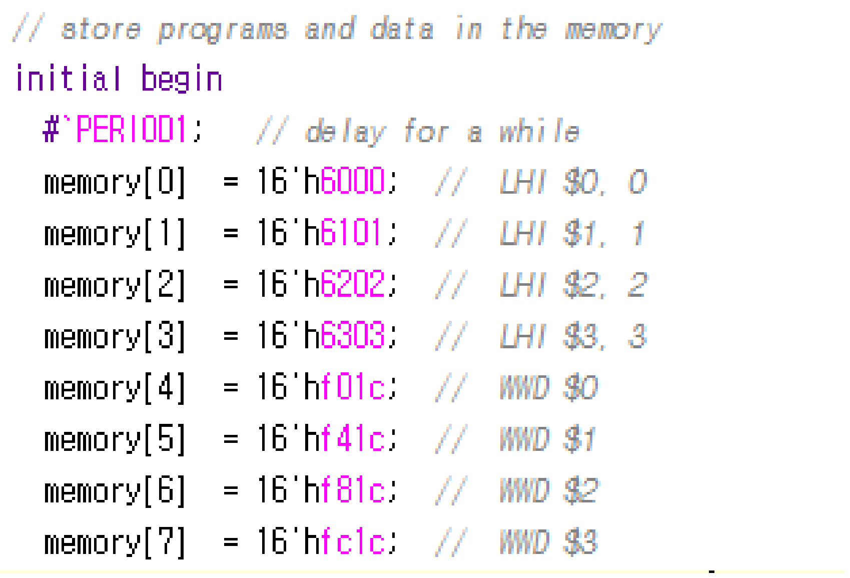
Testbench: Memory model

data == outputData only when *readM* is 1.

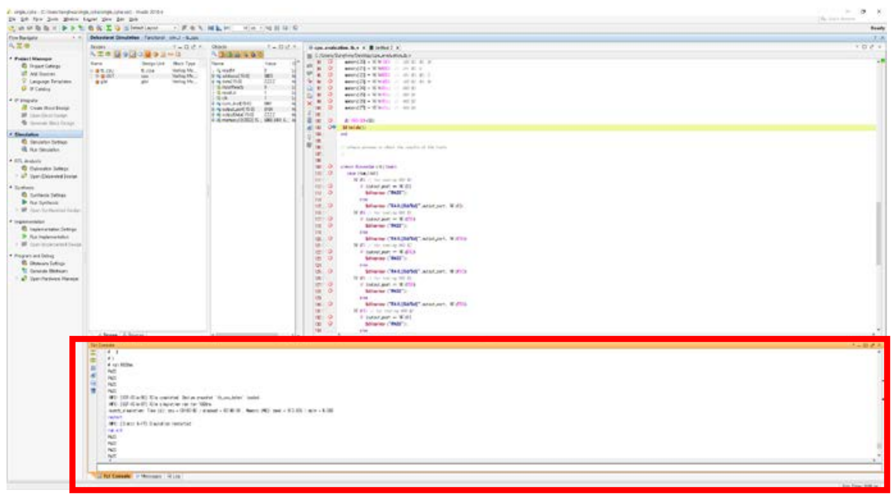


```
assign data = readM ? outputData : `WORD_SIZE'bz;
always begin
  outputData = `WORD_SIZE'bz;
  #`PERIOD1;
  forever begin
    wait (readM == 1);
    #`READ_DELAY;
    outputData = memory[address];
    inputReady = 1;
    //$display("readM: %d, data: %d", outputData, data);
    #(`STABLE_TIME);
    outputData = `WORD_SIZE'bz;
    inputReady = 0;
  end  // of forever loop
end  // of always block for memory read
```

Testbench (In-memory instructions)



```
// store programs and data in the memory
initial begin
  #`PERIOD1;    // delay for a while
  memory[0]  = 16'h6000;  //  LHI $0, 0
  memory[1]  = 16'h6101;  //  LHI $1, 1
  memory[2]  = 16'h6202;  //  LHI $2, 2
  memory[3]  = 16'h6303;  //  LHI $3, 3
  memory[4]  = 16'hf01c;  //  WWD $0
  memory[5]  = 16'hf41c;  //  WWD $1
  memory[6]  = 16'hf81c;  //  WWD $2
  memory[7]  = 16'hfc1c;  //  WWD $3
```



Tcl console

- We provide a testcase code on the eTL board
  - TSC instructions in memory
  - You can modify the code to write your program
  - You can check the result on Tcl console
    - If you pass the testcase, then “PASS” is printed on the console

Thank You