HW #3: Single-Cycle CPU

國立陽明交通大學
NATIONAL YANG MING CHIAO TUNG UNIVERSITY

Chun-Jen Tsai
NYCU
04/11/2024

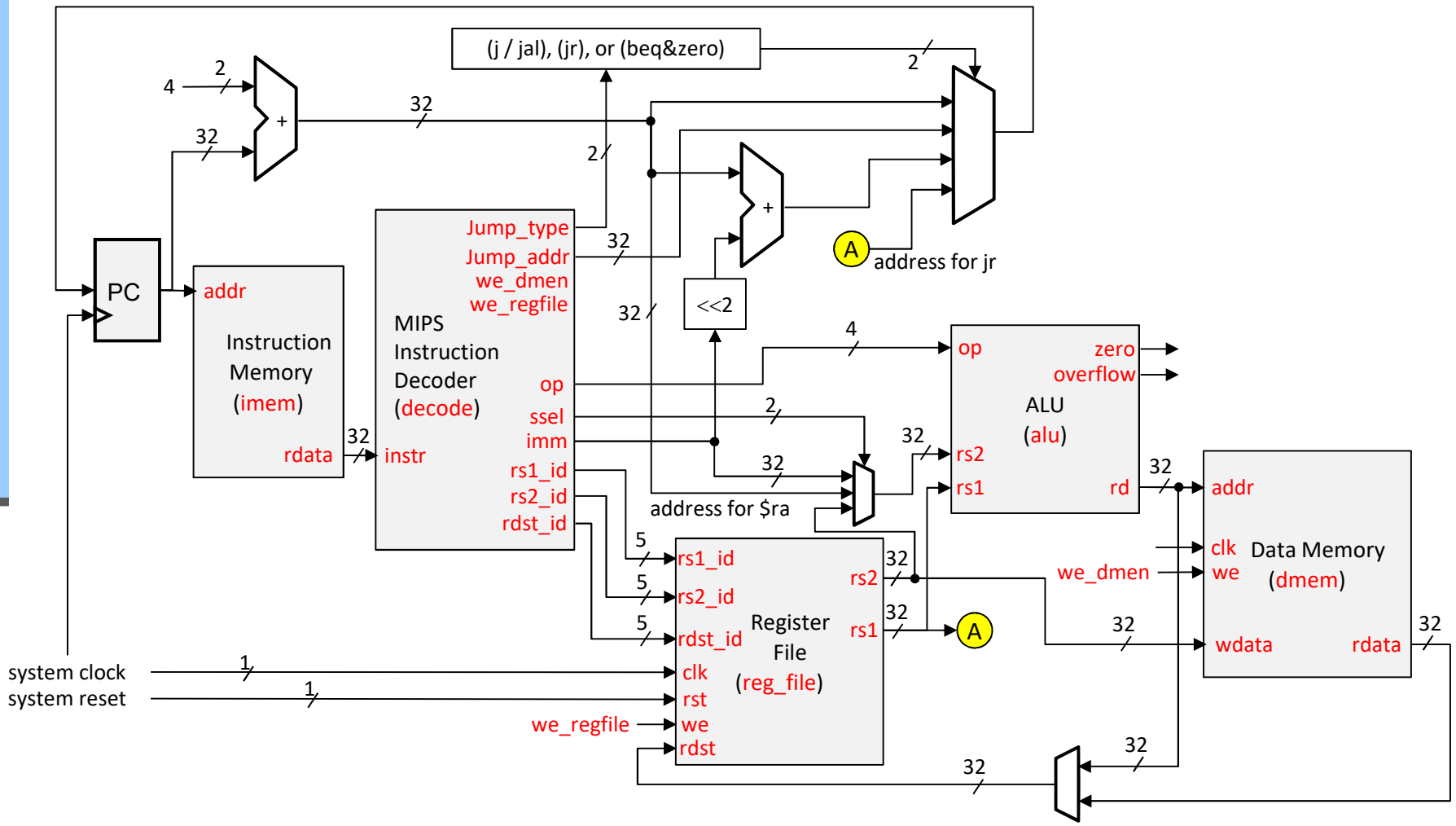
HW #3: Single-Cycle CPU

- Goal: implement a single-cycle MIPS CPU
  - For this HW, you have to add an instruction memory and a data memory to the system of HW #2
  - In addition, you have to add several instructions, including `lw`, `sw`, `j`, `jal`, `jr`, and `beq` to the CPU.

- The deadline of the HW is on 4/25, by 5:00pm.

The Top-Level Block Diagram

- A possible system block diagram is as follows:



The List of Instructions

- Your CPU must support the following instructions:

Assembly instruction	Function	Format	opcode	funct
`add rd, rs, rt`	addition: rd ← rs + rt	R	0x00	0x20
`addi rt, rs, imm`	add immediate: rt ← rs + imm	I	0x08	0x00
`sub rd, rs, rt`	subtraction: rd ← rs - rt	R	0x00	0x22
`and rd, rs, rt`	rd ← rs and rt	R	0x00	0x24
`or rd, rs, rt`	rd ← rs or rt	R	0x00	0x25
`nor rd, rs, rt`	rd ← ~(rs or rt)	R	0x00	0x27
`slt rd, rs, rt`	rd ← (rs < rt)? 32'h1 : 32'h0;	R	0x00	0x2a
`slti rt, rs, imm`	rt ← (rs < imm)? 32'h1 : 32'h0;	I	0x0a	0x00
`lw rt, imm(rs)`	rt←DMEM[rs+imm]	I	0x23	–
`sw rt, imm(rs)`	DMEM[rs+imm] ← rt	I	0x2b	–
`beq rs, rt, imm`	if (rs == rt) PC ← PC + 4 + (imm*4)	I	0x04	–
`jal target_pc`	R[31] ← PC + 4, PC ← {PC[31:28], addr << 2}	J	0x03	–
`jr rs`	PC ← rs	R	0x00	0x08
`j target_pc`	PC ← {PC[31:28], addr << 2}	J	0x02	–

* DMEM[] means data memory, imm is sign-extended to 32-bit before computation

The Instruction Memory (1/2)

- The instruction memory is a combinational word-aligned 32-bit read-only memory that contains 16 words
  - "Word-aligned" means the last 2 bits of the address must be '00'.

```
module imem(
    input  [ 5 : 0] addr,  // byte address
    output [31 : 0] rdata  // read data
);

reg [31 : 0] RAM [15 : 0];

initial // put the machine code of the program here.
begin
    RAM[0]  = 32'h20080017; RAM[1] = 32'h2109002d;
    RAM[2]  = 32'hac090008; RAM[3] = 32'h8c0a0008;
    RAM[4]  = 32'h214bffd3; RAM[5] = 32'h110b0001;
    RAM[6]  = 32'hac080000; RAM[7] = 32'h08000007;
    RAM[8]  = 32'h0; RAM[9]  = 32'h0; RAM[10] = 32'h0;
    RAM[11] = 32'h0; RAM[12] = 32'h0; RAM[13] = 32'h0;
    RAM[14] = 32'h0; RAM[15] = 32'h0;
end

assign rdata = RAM[addr[5:2]];
endmodule
```

The Instruction Memory (2/2)

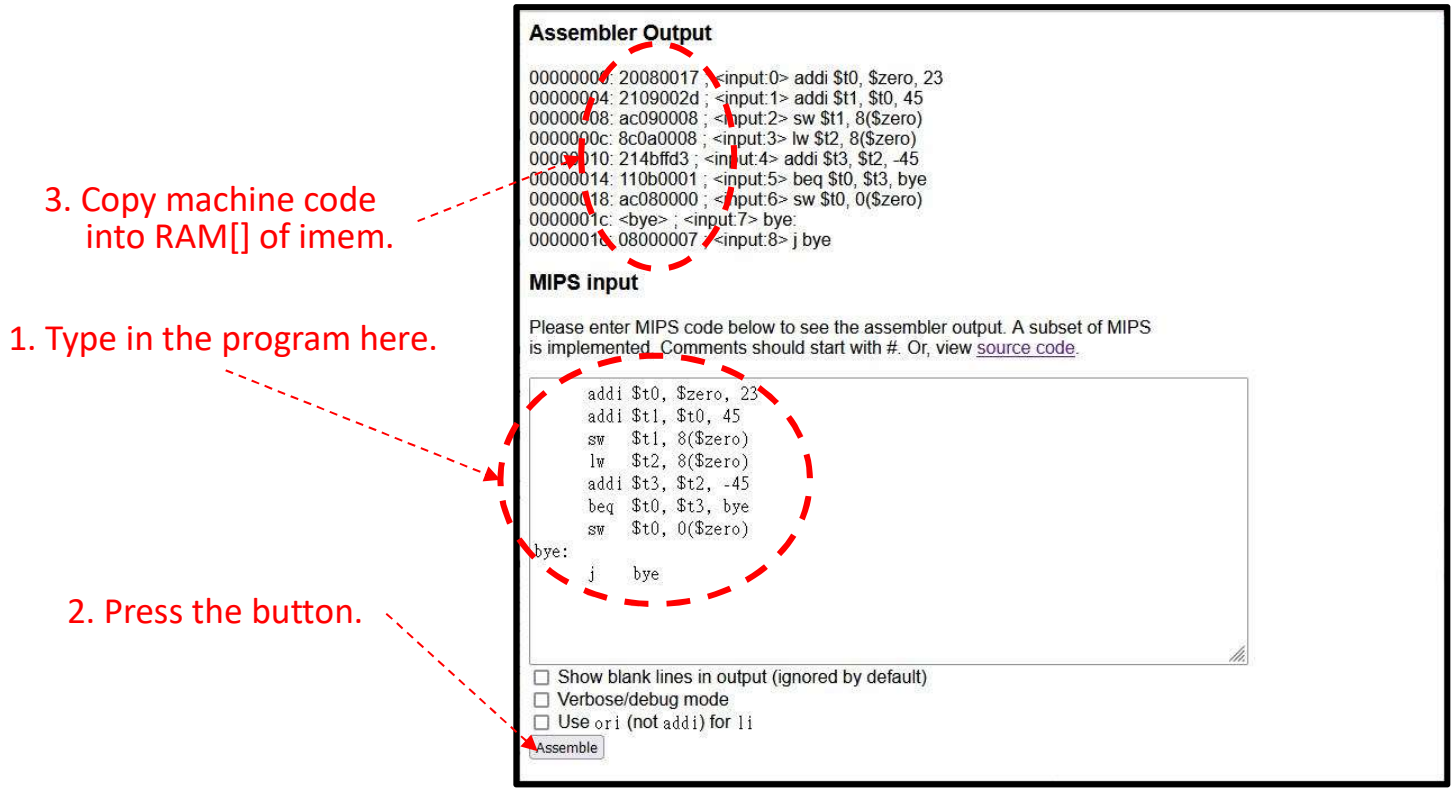
- For example, you can initialize the binary code of the following program into `RAM[]` of `imem` to test your CPU:

```
        addi $t0, $zero, 23
        addi $t1, $t0, 45
        sw   $t1, 8($zero)
        lw   $t2, 8($zero)
        addi $t3, $t2, -45
        beq  $t0, $t3, bye
        sw   $t0, 0($zero)
bye:
        j    bye
```

  - Note: the last instruction of any test programs should contain a jump instruction to itself to lock up the CPU.
  - The machine code of the test program can be obtained using an online assembler discussed in the next slide

An Online MIPS Assembler

- Use the assembler† to generate machine code: https://www.cs.nycu.edu.tw/~cjtsai/mips.php



† The assembler is modified from the project: https://github.com/alanhogan/online-mips-assembler

The Data Memory (1/2)

- The data memory is a sequential word-aligned 32-bit memory that contains 16 words:

```
module dmem(
    input            clk,   // system clock input
    input  [ 5 : 0] addr,  // byte address
    input            we,    // write-enable
    input  [31 : 0] wdata, // write data
    output [31 : 0] rdata  // read data
);

reg [31 : 0] RAM [15 : 0];
integer idx;

initial
begin
    for (idx = 0; idx < 16; idx = idx+1) RAM[idx] = 32'h0;
end

// Read operation
assign rdata = RAM[addr[5:2]];

// Write operation
always@(posedge clk)
begin
    if (we) RAM[addr[5:2]] = wdata;
end
endmodule
```

The Data Memory (2/2)

- For single-cycle CPU, only the PC register and the register file must be implemented as sequential logic
- However, a combinational data memory is composed of latches with an enable signal
  - Latches are typically not used for logic synthesis
- Thus, in this HW, we use a sequential data memory
  - The memory will be initialized to all zeros at start-up

Guidelines for HW#3

- Your top-level block diagram shall be instantiated using a Verilog module `core_top.v`
  - Sample templates of `core_top.v`, `imem.v`, `dman.v`, and the testbench `hw3_tb.v` is available on E3.
  - Do not modify the module interface of `core_top`. TAs will use a different testbench to test your `core_top.v`.
  - The block-diagram synthesized by your `core_top.v` does not have to be exactly the same as the one shown in page 3.
- The `decode` module is the controller in the textbook
  - Its I/O ports shown on page 3 are not complete!
  - You are free to add more ports based on your design

HW #3 Grading Guide

- You should upload all your code to E3 by the deadline
- To evaluate your design, the TAs will use different programs to test your CPU.
  - The program will do some computations and write the result to the data memory
  - At the end of execution, the data memory and the registers should contain the correct result