CA Final Project - Report

B07202020 Hao-Chien Wang
B07901135 Guo-Wei Ho

July 2, 2020

1 Design

Our design is drawn in figure 1, which is mostly identical to the design in the textbook, but a few modules, signals and wires are added to support more instructions.

1.1 The Control Unit

The control unit reads the `opcode`, and output 9 signals:

- `Regwrite`: 1-bit signal.`1` if writing to register is required, `0` otherwise.
- `ALUSrc`: 2-bit signal, one for each ALU input. The input will be from the register if the signal is `0`. The first input will be from PC, and the other input will be from the `ImmGen` if the corresponding signals are `1`.
- `ALUOp`: 2-bit signal. Indicates the instruction type to help ALU control further decides the operation of ALU. The signals are defined as follows:
  - `00`: always add.
  - `01`: always subtract.
  - `10`: R-type intruction. Operation depends on funct7.
  - `11`: R-type intruction. Operation depends on funct7.
- `RegSrc`: 2-bit signal. Together with `ALURegSrc`, they determine the source written back to the register. The meanings of each signals are:

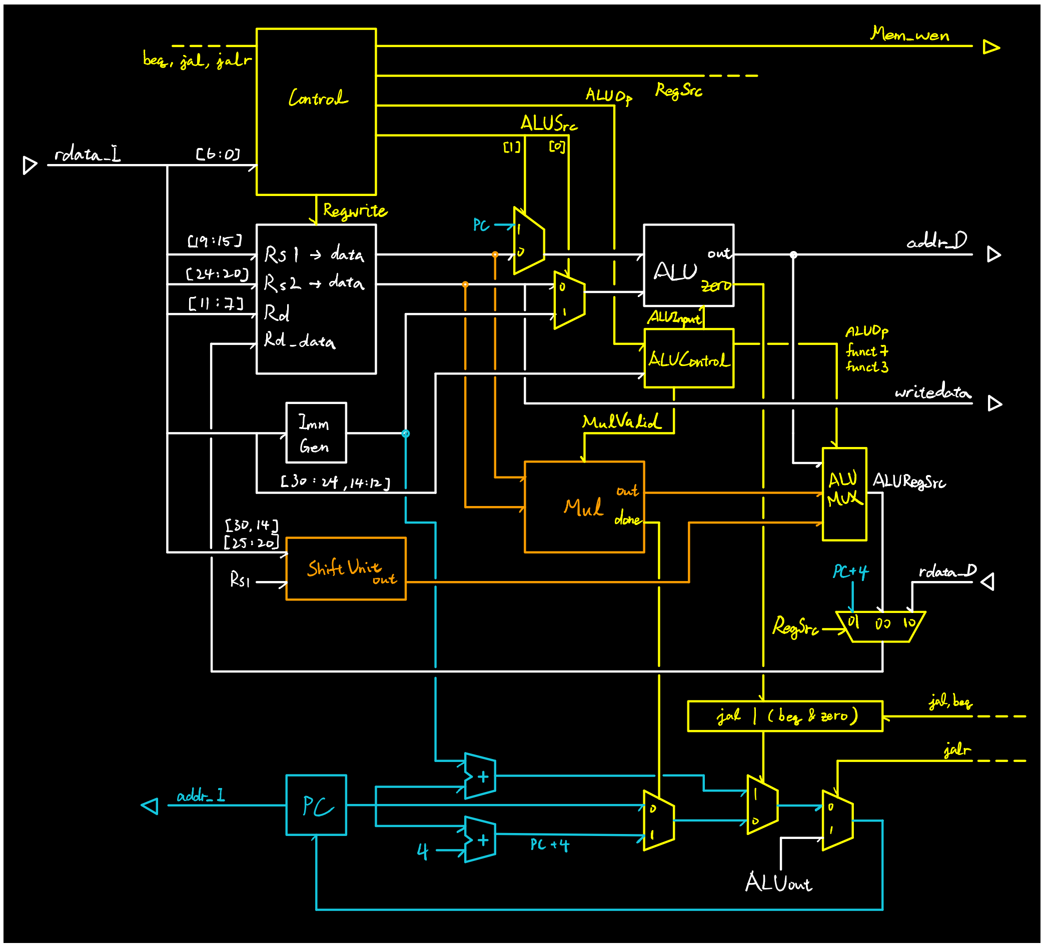

Figure 1: CPU design

  - `00`: ALU
  - `01`: PC + 4
  - `10`: Memory

- `Mem_wen`: 1-bit signal, `1` for writing the memory, `0` otherwise.
- `beq`: 1-bit signal, signaling the branch instruction.
- `jal`: 1-bit signal, signaling the `jal` instruction.
- `jalr`: 1-bit signal, signaling the `jalr` instruction.

The signals for each instructions are listed in table 1.

Intruction	opcode	men_wen_D	RegSrc	ALUop	ALUSrc	Regwrite
lw	0000011	0	10	00	01	1
addi	0010011	0	00	11	01	1
slli	0010011	0	00	11	01	1
slti	0010011	0	00	11	01	1
srai	0010011	0	00	11	01	1
auipc	0010111	0	00	00	11	1
sw	0100011	1	X	00	01	0
add	0110011	0	00	10	00	1
sub	0110011	0	00	10	00	1
mul	0110011	0	00	10	00	1
beq	1100011	0	X	01	00	0
jalr	1100111	0	01	00	01	1
jal	1101111	0	01	X	X	1
DEFAULT		0	000	00	00	0

Table 1: Control signals of each instructions.

1.2 The ALU Control

The ALU control decides the operation of the ALU, and also plays a part in determining the source written back to the register. This module takes `ALUOP`, `funct3` and `funct7` and several output sources as inputs (explained below in the `ALURegSrc` part), then output `ALUInput` to the ALU to control the operation of the ALU, `ALURegSrc` to the multiplexer determining the source written back, and `MulValid` to trigger the multiplication.

- `ALUInput`: 3-bit signal. Determines the operation of the ALU.
  - `010`: add
  - `110`: subtract

  Note that we use 3 bits instead of one, even though there are really two options. The advantage is that it can easily be modified to support more operatios such as `and`, `or`, etc.

- `ALUMUX`: Takes the results of ALU, Multiplier, and Shift Unit, and assign the desired result to `ALURegSrc`. Note that we implement this unit along with `ALUInput` and consider it a part of `ALUControl` since the output also depends on `ALUOp`, `funct3`, and `funct7`.
- `ALURegSrc`: 32-bit data bus. The result from ALU, Multiplier, or Shift Unit to `rd_data` (including the sign bit for the instruction `STLI`)
- `MulValid`: 1-bit signal. `1` if the intruction is `mul`, 0 otherwise. The details about multiplication will be described in section 1.3.

1.3 Multiplication

The mulitiplication module is from HW3. The output signal `done` is `1` unless `MulValid` is `1` and the multiplication state is not in `S_DONE`. `done` controls the PC. When `done` is `0`, the PC is frozen (the next state of PC is the same as the present state). Otherwise it is PC + 4 as usual. This means when instruction is `mul`, the ALU control unit set `mulValid` to 1, and the multiplication process is triggered. During the process the `done` signal is `0` until the result is ready. Before that the PC is frozen, which means all the modules are also frozen except the multiplication module. When the result is ready, the state is in `S_DONE` and `done` is set back to 1, and PC goes on to the next instruction.

2 Instruction Implementation/Datapath

The Implementation and datapath of some instructions. Some instructions are not mentioned since they are already included in the lectures or the section above (`mul`).

2.1 `AUIPC`

To perform the addition of `PC` and `imm`, We added a MUX and an extra bit to the control signal `ALUSrc` so that the first operand can be either `rs1` or `PC`.

2.2 JAL/JALR

For JAL and JALR, the address of the next instruction PC+4 have to be stored to rd. We included this feature in the control signal RegSrc so that rd_data can select PC+4 as source.
For JAL, the target address (PC+imm) is calculated in the extra adder previously designed for BEQ, and will be selected by the control signal jal.
For JALR the target address is calculated in ALU and will be selected by the control signal jalr.

2.3 SLLI/SRAI

We added a shift unit to implement these instructions. It reads the parameters (shift amount, shift direction, ...) from the 32-bit instruction directly, and performs the shift operation on rs1_data. The result will be selected by ALUContol and send to ALURegSrc when needed.

2.4 SLTI

When this instructions is read, ALUControl will use the MSB (sign-bit) of the result of subtraction as ALURegSrc. For example, when rs1 is less than rs2, the result of subtraction will be negative, so 32'b1 will be sent to rd_data through ALURegSrc.

3 Required Screenshots

```
============================================================

Success!
The test result is .....PASS :)

============================================================

Simulation complete via $finish(1) at time 255 NS + 0
./Final_tb.v:173             $finish;
ncsim> exit
```

Figure 2: Simulation time of *leaf example*.

```
============================================================

Success!
The test result is .....PASS :)

============================================================

Simulation complete via $finish(1) at time 4795 NS + 0
./Final_tb.v:173             $finish;
ncsim> exit
```

Figure 3: Simulation time of *fact*.

```
================================================================

Success!
The test result is .....PASS :)

================================================================

Simulation complete via $finish(1) at time 575 NS + 0
./Final_tb.v:173              $finish;
ncsim> exit
```

Figure 4: Simulation time of *HW1*.

```
==========================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
==========================================================================
|     shreg_r_reg     | Flip-flop |  64   |  Y  | N  | Y  | N  | N  | N  | N  |
|    alu_in_r_reg     | Flip-flop |  32   |  Y  | N  | Y  | N  | N  | N  | N  |
|     state_r_reg     | Flip-flop |   2   |  Y  | N  | Y  | N  | N  | N  | N  |
|      cnt_r_reg      | Flip-flop |   5   |  Y  | N  | Y  | N  | N  | N  | N  |
==========================================================================
```

Figure 5: Coding style check.

4 Work Distribution

- Hao-Chien Wang: Control, ALU, ALUControl, debug, report.
- Guo-Wei Ho: Design, instruction generation, ImmGen, multiplier (HW3), multiplexers, all other little stuff, debug.