Documentation of CPU project

Mahdi Haghverdi
Seyed Hussein Husseini
Sina Rabiei

May 15, 2023

Contents

Chapter 1

ISA

In this chapter we will introduce the ISA of our CPU. Many different and important aspects of the ISA will be covered. Covered topics are:

- ISA Table
- ISA Instruction Formats
- ISA Encoding

1.1 ISA Table

The given ISA to our group consists of 12 commands, spread into arithmetic, logical, branching and memory categories.

Table 1.1: ISA table

Type	Assembly Instruction
Arithmetic	`add, sub, mul`
Logical	`and, or, nor, nand, andi, ori`
Branching	`bnq`
Memory	`lw, sw`

Do note that in our CPU all ISA instructions mentioned in table 1.1, deal with registers to get and store data; no operation can be dealt with direct access to memory.

1.2 ISA Instruction Formats

In this CPU we have two types of instruction formats: 1. `R-type`[1] 2. `V-type`[2]; and in this section we will go through details of this formats.

These are the the divided assembly instructions of the CPU:

Table 1.2: `r-type` and `v-type` instructions

Type	Assembly Instruction	Count
`R-type`	`add, sub, mul, and, or, nor, nand`	7
`V-type`	`bnq, lw, sw, andi, ori`	5

The instruction format for **recognizing the instruction**[3], looks like this:

Table 1.3: Instruction format of `r-type` and `v-type` instructions

Type	Opcode	Rs	Rt	Rd	Don't Care	Funct
`R`	`000`	`nnnn`	`nnnn`	`nnnn`	`14-bit`	`nnn`
Type	Opcode	Rs	Rt	immediate value		
`V`	`nnn`	`nnnn`	`nnnn`	`21-bit`		

[1]Register format instruction
[2]Value format instruction
[3]encoding the assembly instruction in binary

1.3 ISA Encoding

All CPUs only understand zeros and ones, in order to tell the CPU to do this and to do that, we have to encode the instructions.

The instruction code table is below:

Table 1.4: ISA encoding

Instruction (`r-type`)	Opcode	Funct
`add`	000	001
`sub`	000	010
`mul`	000	011
`and`	000	100
`or`	000	101
`nor`	000	110
`nand`	000	111

Instruction (`v-type`)	Opcode
`bnq`	111
`lw`	001
`sw`	010
`andi`	100
`ori`	101

Chapter 2

Registers

In this chapter we will look at the register file of the CPU and the design decisions of this important part. Covered topics are:

- Number Of Registers
- Naming Conventions of Registers
- Capacity Of Registers

All the operations which take place in the CPU get their operands and store their results in the register file. For example the instruction:

```
add $r1, $r2, $r3
```

Which means `$r1 = $r2 + $r3`, the data are fetched from the registers `$r1,` `$r2` and `$r3`.

There are some decisions for the design of the register file of the CPU which will be discussed below.

2.1 Number Of Registers

After discussions between the team members we decided to put 16 registers in the register file of the CPU.

With this number of registers, we can access them by a four-bit number, because $16 = 2^4$

2.2 Naming Conventions of Registers

The registers are names as follows:

1. `$zero`
2. `$at`
3. `$a1`
4. `$a2`
5. `$a3`
6. `$l1`
7. `$l2`
8. `$l3`
9. `$t1`
10. `$t2`
11. `$t3`
12. `$t4`
13. `$t5`
14. `$t6`
15. `$t7`
16. `$t8`

2.2.1 Register Specifications

These register specifications are done just for convenience and safety of the compiled programs. For example, a logical operation operands are better to be placed in `$l#` registers. This convention will guarantee the data safety during the program execution.

2.2.1.1 `$zero` register

Because the value `0` is used a lot, we put it here. This register will not be overwritten.

2.2.1.2 `$at` register

This is the register provided for the assembler to use on its own.

2.2.1.3 `a#` registers

`a#` registers are named after the word `arithmetic`. These registers are specific for arithmetic operations.

2.2.1.4 `l#` registers

`l#` registers are named after the word `logical`. These registers are specific for arithmetic operations.

2.2.1.5 `t#` registers

`t#` registers are named after the word `temporary`. These registers can be used freely by the compiler or assembly programmer.

2.3 Capacity Of Registers

This CPU's registers are 32-bit registers with the capability of parallel load and read.

Chapter 3

Control Unit

Control Unit and Data Path are the two hearts of all CPUs. In this chapter we will show the design of our CPU's control unit.

Control unit, produces some signals to power on or power off some parts of the data path to control the data flow between the gates, registers, the alu and multiplexers of the data path. The table below shows all the control unit signals we need for our ISA (discussed in ISA chapter) Here is the table of all

3.1 Control Unit Signals Table

There are some zeros and ones in the table Control Unit Signals Table, here are the meanings:

In `RegDst`, `0` means: `Rt` in the instruction and `1` means: `Rd`.

In `ALUSrc`, `0` means: a register and `1` means `immediate value`.

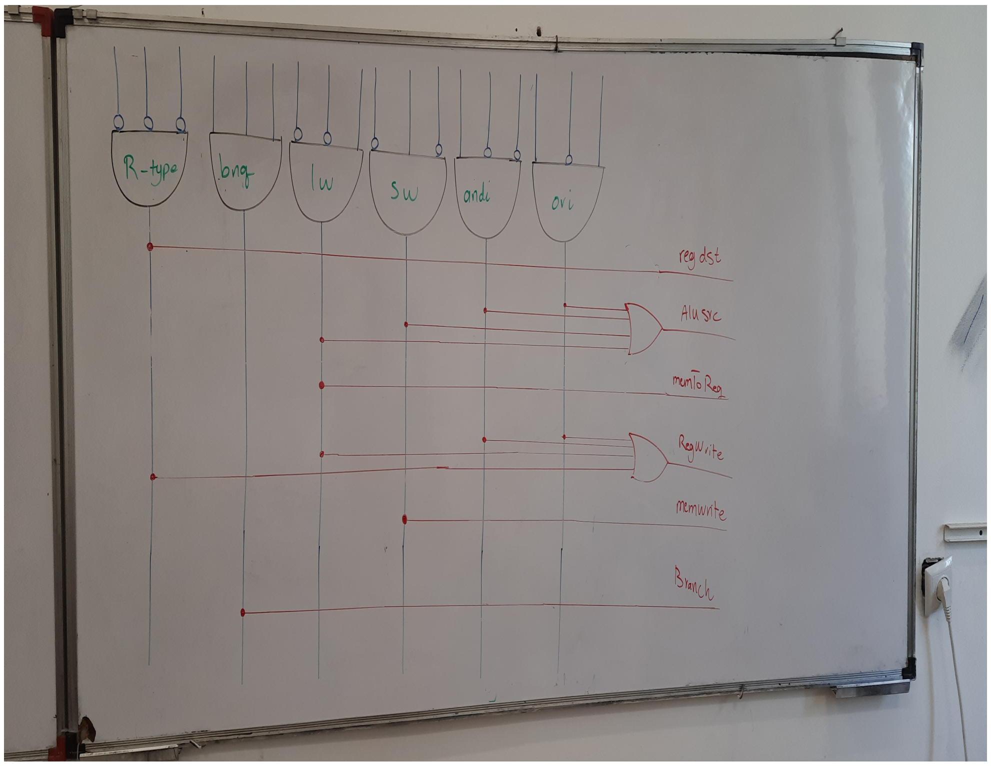
In `MemToReg`, `0` means: `from ALU result to a register` and `1` means `from memory to a register`,

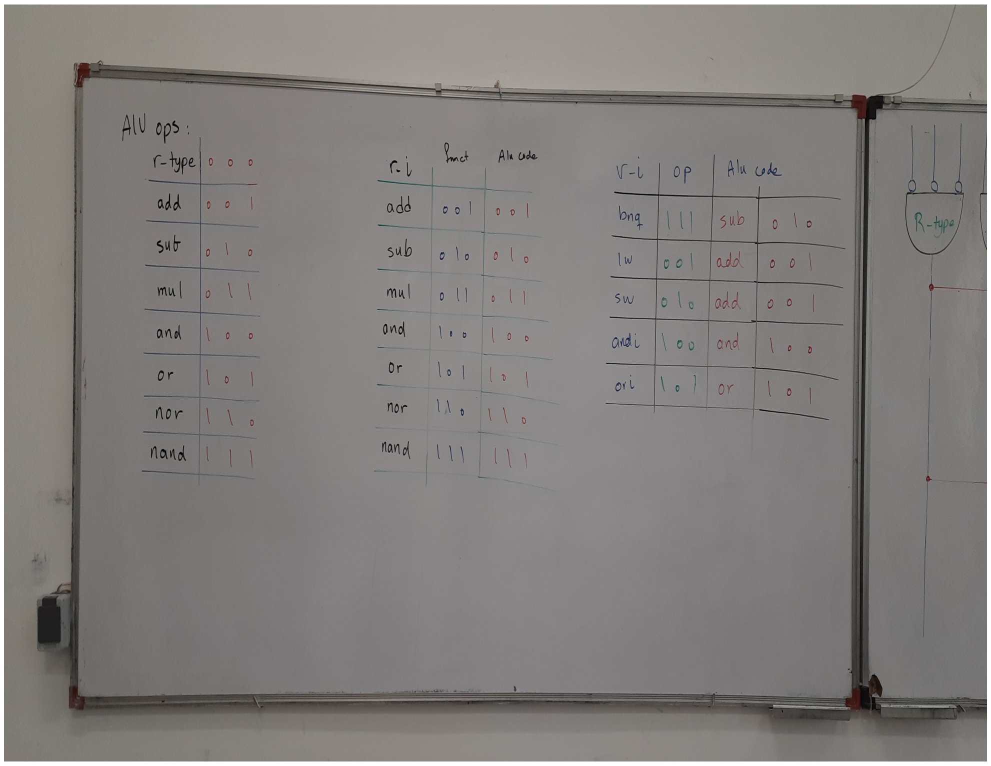
In `RegWrite`, `1` means: write to register from ALU.

In `MemWrite`, `1` means: write to memory from ALU.

In `Branch`, `1` means: put one in the branch `and` gate.

3.2 Control Unit Design





AlU ops :

r-type	0 0 0
add	0 0 1
sub	0 1 0
mul	0 1 1
and	1 0 0
or	1 0 1
nor	1 1 0
nand	1 1 1

r-i	funct	Alu code
add	0 0 1	0 0 1
sub	0 1 0	0 1 0
mul	0 1 1	0 1 1
and	1 0 0	1 0 0
or	1 0 1	1 0 1
nor	1 1 0	1 1 0
nand	1 1 1	1 1 1

r-i	op	Alu code	
bnq	1 1 1	sub	0 1 0
lw	0 0 1	add	0 0 1
sw	0 1 0	add	0 0 1
andi	1 0 0	and	1 0 0
ori	1 0 1	or	1 0 1

Table 3.1: Control Unit Signals

Control Signal	add	sub	mul	and	or	nor	nand	bnq	lw	sw	andi	ori
RegDst	1	1	1	1	1	1	1	x	0	x	0	0
ALUSrc	0	0	0	0	0	0	0	0	1	1	1	1
MemToReg	0	0	0	0	0	0	0	x	1	x	0	0
RegWrite	1	1	1	1	1	1	1	x	1	x	1	1
MemWrite	0	0	0	0	0	0	0	0	0	1	0	0
Branch	0	0	0	0	0	0	0	1	0	0	0	0
ALUctrl	add	sub	mul	and	or	nor	nand	sub	add	add	and	or