DrMIPS y las instrucciones

Juan Heguiabehere

Archivos `.set` (Instrucciones)

Los archivos `.set` tienen dos tipos distintos de información: la de cómo se compila una instrucción en assembly a código máquina, y la de cómo tienen que ser las señales de control para su ejecución.

Ejemplo: `addi`

Para definir la instrucción `addi` se comienza por incluirla en el listado de instrucciones, que es la sección "instructions":

```
"j":    {"type": "J", "args": ["target"], "fields": {"op": 2, "target": "#1"}, "desc": "PC = target"},
"addi": {"type": "I", "args": ["reg", "reg", "int"], "fields": {"op": 8, "rs": "#2", "rt": "#1", "imm": "#3"}, "desc": "$t1 = $t2 + 23"},
"beq":  {"type": "I", "args": ["reg", "reg", "offset"], "fields": {"op": 4, "rs": "#1", "rt": "#2", "imm": "#3"}, "desc": "PC += ($t1 ==
```

En este caso, los campos marcan que la instrucción "addi" es de tipo I, que sus argumentos deben ser interpretados como dos registros más un valor inmediato, que el opcode (para la unidad de control) es 8, y que el registro de origen es el primer argumento, el de destino es el segundo argumento, y que el valor inmediato es el tercer argumento. Los bits de la instrucción en los que esos valores se codifican están en la sección "types", al principio del archivo:

```
"R": [{"id": "op", "size": 6}, {"id": "rs", "size": 5}, {"id": "rt", "size": 5}, {"id": "rd", "size": 5}, {"i
"I": [{"id": "op", "size": 6}, {"id": "rs", "size": 5}, {"id": "rt", "size": 5}, {"id": "imm", "size": 16}],
"J": [{"id": "op", "size": 6}, {"id": "target", "size": 26}]
```

En la sección "control" se especifican las salidas de la unidad de control, según el opcode señalado en la definición de la instrucción:

```
"control": {
        "0": {"RegDst": 1, "RegWrite": 1, "ALUOp": 2, "ALUSrc": 0, "MemToReg": 0},
        "8": {"RegDst": 0, "RegWrite": 1, "ALUOp": 0, "ALUSrc": 1, "MemToReg": 0},
        "2": {"Jump": 1},
        "4": {"ALUOp": 1, "ALUSrc": 0, "Branch": 1},
        "35": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "RegWrite": 1, "MemRead": 1, "MemWrite": 0, "MemToReg": 1},
        "43": {"ALUOp": 0, "ALUSrc": 1, "RegDst": 0, "RegWrite": 0, "MemRead": 0, "MemWrite": 1, "MemToReg": 0}
},
```

Aquí vemos que cuando el opcode es 8, se deben setear RegWrite y ALUSrc en 1, RegDst y MemToReg en 0, y ALUOp en 0.

En la sección "alu", finalmente, se establece qué operación de la ALU se debe ejecutar, teniendo en cuenta "aluop" (que es el ALUOp de la sección de control) y el campo "func" si es necesario:

```
"alu": {
        "aluop_size": 2,
        "func_size": 6,
        "control_size": 4,
        "control": [
                {"aluop": 0, "out": {"Operation": 2}},
                {"aluop": 1, "out": {"Operation": 6}},
                {"aluop": 2, "func": 32, "out": {"Operation": 2}},
                {"aluop": 2, "func": 34, "out": {"Operation": 6}},
                {"aluop": 2, "func": 36, "out": {"Operation": 0}},
                {"aluop": 2, "func": 37, "out": {"Operation": 1}},
                {"aluop": 2, "func": 39, "out": {"Operation": 12}},
                {"aluop": 2, "func": 42, "out": {"Operation": 7}}
        ],
        "operations": {
                "0": "and",
                "1": "or",
                "2": "add",
                "6": "sub",
                "7": "slt",
                "12": "nor"
        }
}
```

Finalmente, en la sección "operations" de la ALU, tenemos las operaciones a las que hacen referencia los códigos de operación marcados en "control" de la ALU. En este caso, la operación 2, afortunadamente, es `add`.