ELEC-5200

Computer Architecture

FROM: Jacob Howard

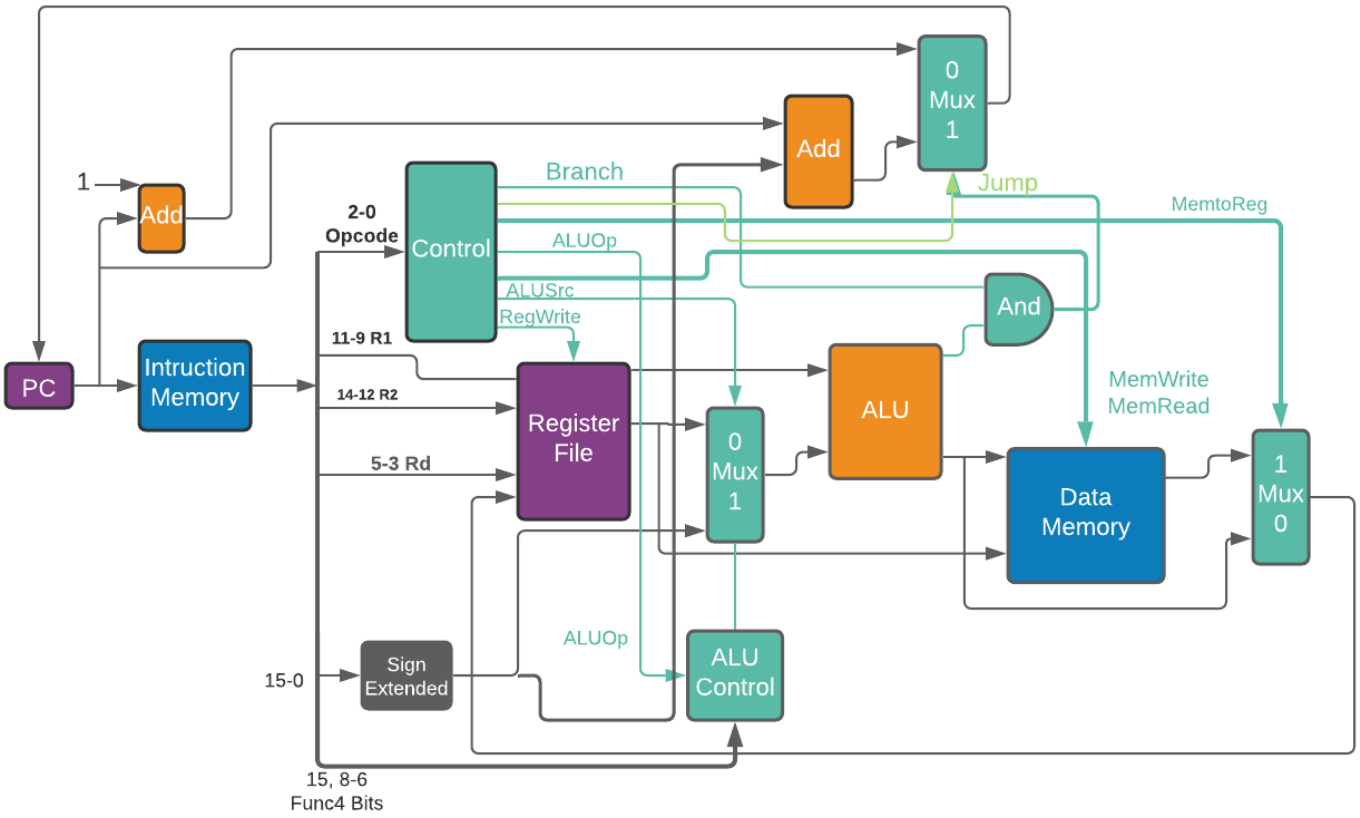
TO: Dr. Harris & Tucker Johnston

DUE DATE: 10/8/21

CPU Design – Part 2

Datapath

This is the Datapath design I have created. The design is a single-cycle datapath as I felt the design is easier to understand and I will run into less issues on future designs. The main components of the design concist of a PC, Intruction and Data type Memory, a Control Path, a Register File, and an ALU. Also, I use a sign extended block to deal with positive and negative immediate numbers. (Note that bit 15 and bits 8-6 are used for Func4 bits, but the 4th bit is always zero for my ISA design. So, we could just write that line to read bits 6-6 and not worry about the 15th bit on r-type instructions)



ISA Review

Below in *Table 1* is the ISA's format I had Developed in Part 1 of the CPU Project. This chart will help one understand the design choices on where each bit of the 16-bit word instructions go.

Type	15th bit	14-12	11-9	8-6	5-3	2-0 bits
R-Type	Funct4[3]	R2	R1	Func4[2:0]	Rd	opcode
I-Tpye	Imm[3:0]		R1	Func4[2:0]	rd	opcode
S-Type	Imm[3]	R2	R1	Func4[2:0]	Imm[2:0]	opcode
SB-Type	Imm[3]	R2	R1	Func4[2:0]	Imm[2:0]	opcode
UJ-Type	Imm[9:0]				rd	opcode

Table 1

Datapath Description

Firstly, the design has a Program Counter (PC). The PC is used as a pointer to the next instruction. The PC can be told to move up one instruction after an instruction is finished by adding 1 to the PC or branch forward/backwards a certain amount of instructions.

Next the design has Instruction Memory. The PC will point to specific lines of the memory where instructions are stored. This is where instructions are stored and read from.

Once the Instruction Memory is read and produces an 16-bit word instruction, the Control Unit is what interprets what type of instruction is being performed by reading its opcode (the first 3 bits in the 16-bit word instruction). The Control Unit decides what will be done with the 16-bit word instruction based on its opcode. It could branch, do arithmetic, write to register file, read/write from memory, etc. A component I added to the control unit in my design is the ability to account for uj-type instructions (Unconditional jumps). If the opcode for a jump is read, the jump skips the and-gate (which is used for comparisions on branches) and goes straight to the mux to declare a jump. The sign extended immediate is added to the pc ad jumps forwards/backwards the designed amount.

The register file will be written to based on the control unit. If it will be written to, it takes the corresponding bits of Reg 1, Reg 2, Reg D, and/or Data Memory based on the instruction.

The Register file privides data to the ALU. The ALU is responsible for all arithmetic type instructions. It can write data to memory and/or check if instructions should branch.

Data Memory is used to read/write data based on what is decided by the control unit. Also, note that sign extended block is used for immediate positive and negative numbers for branching, jumping, and immediate arithmetic.

Datapath Truth Tables

The two truth tables required to understand how the datapath design works are truth tables for the Control Unit and the ALU. The Control Unit is important to understand as it the the decision maker on what type of instruction must be performed. Along with the Datapath, if an arithmetic instruction is required, the ALU must read the Function-4 code in order to decide which arithmetic instruction must be performed. The ALU is also used for comparasins for branch functions.

The only thing missing from the control unit is the ability to interpret a halt instruction. I am unsure of how to implement that as there was nothing from class or the notes that explained hot to implement a halt in the Control Logic. There also may be some bugs or incorrect datapath links that I missed that the TA or professor might catch. So, there could be adjustments to the final design of the Datapath

Control Unit Logic

Type	Opcode			Outputs: Control Signals								
----	2	1	0	ALUSrc	MemToReg	RegWrite	Mem Reed	Mem Write	Branch	ALUOp 1	ALUOp 2	Jump
r	0	0	0	0	0	1	0	0	0	1	0	0
i	0	0	1	0	0	1	0	0	0	0	1	0
lw	0	1	0	1	1	1	1	0	0	0	0	0
sw	0	1	1	1	X	0	0	1	0	0	0	0
sb	1	0	0	0	0	1	0	0	1	0	1	0
uj	1	0	1	0	0	1	0	0	0	0	1	1

ALU Control Truth Table

Intruction Opcode	ALUOp	Func4	ALU Decision
lw	00	xxxx	add
sw	00	xxxx	add
Sb (beq)	01	0000	sub
Sb (bgt)	01	0001	sub
Sb (bge)	01	0010	sub
Sb (blt)	01	0011	sub
r-type	10	0000	add
r-type	10	0001	sub
r-type	10	0010	and
r-type	10	0011	or
r-type	10	0100	xor
r-type	10	0101	shift left
r-type	10	0110	shift right

i-type	01	0000	addi
i-type	01	0001	subi
i-type	01	0010	ori
i-type	01	0011	xori
i-type	01	0100	sli
i-type	01	0101	sri

Design Overview & Conclusion

The tradeoffs for choosing a single-cycle datapath design is that the overall performance will be slower. The reason I chose single-cycle is because the overall design is simpler, leaving less room for errors in the future when implementing later designs. Also, the single-cycle datapath is easier to understand, so troubleshooting should be less intensive when encountering problems. As of now, edge-triggered registers seems to be the choice in design as it seems to be simpiler to implement but this could change in the future.

Again, the only thing I am aware of that is missing from the control unit is the ability to interpret a halt instruction. I am unsure of how to implement that as there was nothing from class or the notes that explained a way to implement a halt in the Control Logic. There also may be some bugs, incorrect datapath links, or other missing this that I missed that the TA or professor might catch. So, there could be adjustments to the final design of the Datapath.