ELEC-5200

Computer Architecture

FROM: Jacob Howard

TO: Dr. Harris & Tucker Johnston

DUE DATE: 9/21/21

CPU Design – Part 1

ISA

This is the ISA design I have created. It consists of 8 different 3-bit opcodes used for determining type and instruction. Function3 is a 3-bit code to determine specific instructions once the type has been determined from the opcode. Currently Funct1 is not necessaly being used (this could be used later on in design or adjusted to give Funct3 a 4-bit code, instead of 3). Funct1 ad been set to 0 for all R-types currently. Registers have been set to 3 bit, allowing only 8 different registers for design.

Operation	Format	Opcode (3-bits)	Destination Reg. (3-bits)	Source Reg. (3-bits)	Target Reg. (3-bits)	F4	Description
Arithmatic Instructions							
add	R	000	Rd	Rs_1	Rs_2	0000	*Adds: Rd = Rs+Rt*
sub	R	000	Rd	Rs_1	Rs_2	0001	*Subtracts: Rd = Rs - Rt*
and	R	000	Rd	Rs_1	Rs_2	0010	*Logic AND: Rd = Rs&Rt*
or	R	000	Rd	Rs_1	Rs_2	0011	*Logic OR: Rd = Rs|Rt*
xor	R	000	Rd	Rs_1	Rs_2	0100	*Logic XOR: Rd = Rs^Rt*
sl	R	000	Rd	Rs_1	----	0101	*Shifts left: Rd = Rs<<1*
sr	R	000	Rd	Rs_1	----	0110	*Shifts right: RD = Rs>>1*
Immediate Arithmatic (extra un-unique instructions)							
addi	i	001	Rd	Rs_1	*Immidiate*	0000	*Add Immediate: Rd = Rs+imm*
andi	i	001	Rd	Rs_1	*Immidiate*	0001	*Logic AND immediate: Rd = Rs&imm*
ori	i	001	Rd	Rs_1	*Immidiate*	0010	*Logic OR immediate: Rd = Rs|imm*
xori	i	001	Rd	Rs_1	*Immidiate*	0011	*Logic XOR immediate: Rd = Rs^imm*
sli	i	001	Rd	Rs_1	*Immidiate*	0100	*Shift left immediate:*
sri	i	001	Rd	Rs_1	*Immidiate*	0101	*Shift right immediate:*

Load/store instructions							
lw	i	010	Rd	Rs_1	*Immidiate*	0000	*Load word: Rd = Rs1+imm*
sw	s	011	----	Rs_1	Rs_2	0000	*Store word: M[R[Rs1]+imm](15:0)=[Rs2](15:0)*
Branches/Jumps							
operation	**Type: sb**	**Opcode 3-bits**	**3-bits**	**3-bits**	***4-bits***	**F4**	***Description***
beq	sb	100	Rs_1	Rs_2	*immidiate*	0000	*Branch if equal to: if(Rs==Rt)PC=PC+imm*
bgt	sb	100	Rs_1	Rs_2	*immidiate*	0001	*Branch if greater than: If(Rs_1>Rs_2)PC=PC+imm*
bge	sb	100	Rs_1	Rs_2	*immidiate*	0010	*Branch if greater than or equal to: If(Rs1>=Rs2)PC=PC+imm*
blt	sb	100	Rs_1	Rs_2	*immidiate*	0011	*Branch if less than: If(Rs1<Rs2)PC=PC+imm*
Operation	Type: uj	Opcode 3-bits	3-bits	10-bis	*Description*		
jal	uj	101	Rd	Immediate	*Jump and link: Rd=PC+4;PC=Rs1+imm*		
Operation	**Format**	**Opcode (3-bits)**	**Destination Reg. (3-bits)**	**Source Reg. (3-bits)**	**Target Reg. (3-bits)**	**F4**	**Description**
jalr	i	110	Rd	Rs_1	*immidiate*	0000	*Jump and link register: Rd=PC+4;PC=Rs1+imm*
Halt							
halt	-	111	-	-	-	0000	*Halts the processor*

There are only 8 unique opcode instructions. The opcode will be used to determine instruction or type. I also use a 3-bit function to determine instructions for type-opcodes

Instuction Format

Below in *Table 1* is the instruction format for my ISA. The instruction format shows how the opcode will be used to parse different “types”. The types being used in my ISA design are: Regsiter, Immediate, Store/Branch, andUnconditional Jump.

I chose to use an opcode and register size of 3-bits. This allows for a function field of 4-bits to determine instructions. This was done to determine the type with opcode and the instruction with the function field. This may also allow for more instructions to be added to the “16-bit word” length of machine code the project specifies.

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

This table is a compressed “16-bit” version of the RISC-V “32-bit” core instruction format, shown below in *Figure 2.*

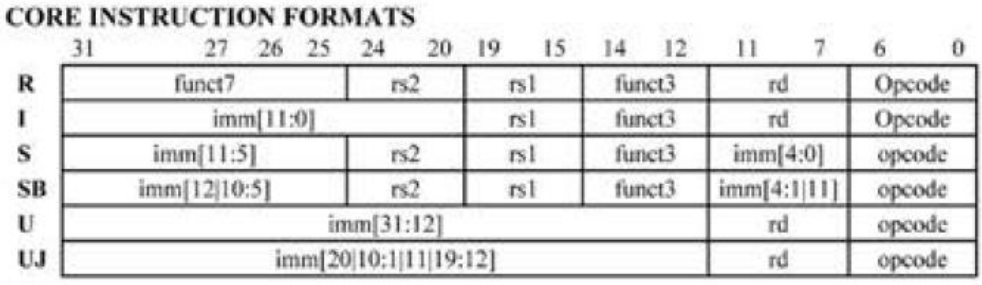

Figure 2

Registers

Since I only allowed for 3-bits for registers, this means I can only have 8 Registers. For register design, I have suggested what each registers purpose should be, but this is not a permanent fix to regiter usage. The table below is a suggestion on what to use for each register.

Reg #	Use
x0	Permantely set zero (for comparisons)
x1	Return Addr.
x2	SP
x3	General Purpose Reg
x4	General Purpose Reg
x5	Function Argument
x6	Function Argument
X7	Return Value

Resister Function Table

ISA Justification

For my ISA design, many of the instructions I included were to follow the design specifications of Part 1 of the CPU Design Project. I opped to add immediate arithmetic instructions to make some problem solving faster/easier. Some logic instructions were added for various tasks, and jumps and branches were added to account for various thinks, like loops for example. I chose for 3-bit opcodes as a way to determine type/instruction. Since I chose a 3-bit opcode, I will have no more than 8 unique opcode instructions for my design. This is under the "16 unique instructions" max limit requested by the assignment.

With my opcode and registers at 3-bits, this gives me room to add more instructions to my ISA without going over the 16 unique instructions limit. I use a 4-bit function do determine exact instructions once the opcode has been read. For example, all R-Type instructions have an opcode of "000" which will determine correct parsing, and the 4-bit function code will be what determins what instruction is used.

In my design, it is also possible to leave out certain branch instructions. You could only have greater than and equal to branch instructions or less than and equal to instructions. While it may be easier to have both types, it is not needed. For now, some of both types have been listed in the ISA since I have not gone over 16 unique instructions.

I am also considering adding a "not" function, certain branch functions (like bne), and load immediate (li) to ISA design to simplify assembly. Adding Logic NOT may make it simpiler to turn numbers negative.

C Construct Examples

I will be showing software examples written in C and assembled in my ISA design here. Each example will show different ways the ISA would certain algorithms written in the C software language. The examples will not be fully written out C code or Assembly code, but will give a general idea on how the ISA I designed will work for required functions.

Simple Add/subtract

C code *(assume sum is in reg x3, i in x4, and j in x5)* **ISA**

sum = (I + j) - 1;	Add x5, x5, x4 #adds I and j and stores in x5 Addi x3, x5, -1 #subtracts 1 from x5 and stores in x3

Ex 1 (sub instruction may not have to be used)

Simple for Loop

C code *(assume x3 = 0, x4 = 4)* **ISA**

for (int i = 0, i > 3) {
i++ //loops until I = 3
}
	Loop:
	Addi x3, x3, +1 #adds 1 to x3 and stores in x3
	Blt x3, x4, Loop #compares reg x3 and x4, if less than, loops
	Jal Exit #if x3 is greater than x4, jumps to exit
	
	Exit:
	Halt #stops processor

Ex 2

Simple if/else

C code *(assume x3 = 0, x4 = 3)* **ISA**

if (int i = 0, i > 3) {
i++ //adds 1
}
Else{
i-- //sub 1
}
	Bgt x3, x4, Loop #compares reg x3 and x4, if greater than, goes to if condition
	Jal Else #if x3 is less than x4, jump to else
	
	If:
	Addi x3,x3, +1 #adds 1 to x3 and stores in x3
	Jal exit #jumps to exit once complete
	
	Else:
	Addi x3, x3, -1 #sub 1 to x3 and stores in x3
	
	Exit:
	Halt #stops processor

Ex 2