Project #2 ISA Design Report for ECE 366

Professor Wenjing Rao

Group #11

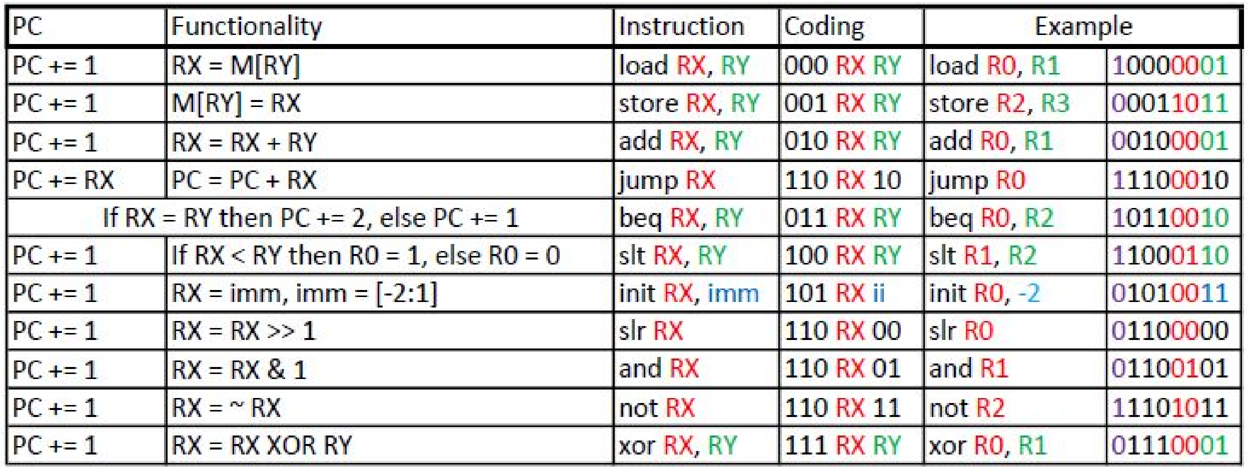
Teja Thangella
Cameron Sprouse
Zachary Szczesniak

Teja Thangella
Zachary Szczesniak
Cameron Sprouse
Group #11
ECE 366
Project #2 Report

Part A:

1. The name of our Instruction Set Architecture is CTZ because it is the first letter of each creator's first name. The initial goal of our ISA was to support the two required programs using only a 7-bit instruction with an even parity bit. These two conditions limited the number of instruction and registers that could be used. The first step of the design process was to write the programs in MIPS, minimizing the number of commands, registers, and instruction count. This allowed us to determine, which commands are essential for the programs. Next, we determined an encoding scheme for the ISA design based on that selection of commands. The overall goal of the ISA was to complete the two required programs with the minimum instruction count. This was achieved by changing how our branch and jump commands executed allowing for more efficient branching and flow control.

2. Our ISA supports 11 instructions below is the encoding scheme.



PC	Functionality	Instruction	Coding	Example	
PC += 1	RX = M[RY]	load RX, RY	000 RX RY	load R0, R1	10000001
PC += 1	M[RY] = RX	store RX, RY	001 RX RY	store R2, R3	00011011
PC += 1	RX = RX + RY	add RX, RY	010 RX RY	add R0, R1	00100001
PC += RX	PC = PC + RX	jump RX	110 RX 10	jump R0	11100010
If RX = RY then PC += 2, else PC += 1		beq RX, RY	011 RX RY	beq R0, R2	10110010
PC += 1	If RX < RY then R0 = 1, else R0 = 0	slt RX, RY	100 RX RY	slt R1, R2	11000110
PC += 1	RX = imm, imm = [-2:1]	init RX, imm	101 RX ii	init R0, -2	01010011
PC += 1	RX = RX >> 1	slr RX	110 RX 00	slr R0	01100000
PC += 1	RX = RX & 1	and RX	110 RX 01	and R1	01100101
PC += 1	RX = ~ RX	not RX	110 RX 11	not R2	11101011
PC += 1	RX = RX XOR RY	xor RX, RY	111 RX RY	xor R0, R1	01110001

3. Four general purpose registers that are supported by the CTZ (R0, R1, R2, R3). R0 is used as the flag register for the boolean output (0,1) of the set less than instruction (slt).

4. CTZ uses the single branch instruction beq. The beq command is unique for our ISA as the beq command adds pc by 2 if the statement is satisfied and 1 if not. The branch distance for the command would be 2. An example of this would be:

Assembly Code	Machine Code
beq R1,R0 #R1=1,R0=1	011 01 00
add R1,R1 #R1=2	010 01 01
add R0,R0 #R0=2	010 01 01

The result would be R0=2

The jump instruction is also used for flow control. Jump modifies PC by adding the value RX to PC. The jump distance for this command is −32,768 to 32,767 as this is the max value that can be stored in a 16 bit register using 2's complement. When used in combination with branch equal the distance that the program can move based on a conditional response grows exponentially.

Assembly Code	Machine Code	Comment
jump R1	0 1101001	# PC = PC + 5 (R1= 5)

5. CTZ uses two addressing modes register-only and immediate addressing. The only instruction to use immediate addressing is init because the value of the register is set to the 2 bit immediate value included in the encoding of the instruction.

Assembly Code	Machine Code	Comment
init R0, -2	0 1010011	# R0 = -2

All other CTZ instructions use register-only addressing for all source and destination operands.

Assembly Code	Machine Code	Comment
load R0, R1	1 0000001	# R0 = M[R1]
add R1, R3	0 0100111	# R1 = R1 + R3

Part B:

1. One of the unique features of our ISA is that it is flexible as it can be used for more than one project and can adapt to various programs. This is done through the use of clever encoding and special hardware designs. Unlike its counterpart, CTZ has more unique commands such as not. This command flips the bits of a number allowing the ISA to support negative numbers. This allowed us to remove the subtract command as it is unnecessary and much easier to create negative numbers and add them together, creating a subtraction function. Another feature that is unique to our ISA is the jump command. This instruction is unique as it does not jump by immediate value or by preselect default, rather by a value in a register this allows for better construction of loops and flow control. As opposed to My_straightforward_ISA that has a very small branch distance while CTZ has practically an infinite branch distance. This also combines well with the branch command that jumps over an exit command for a loop. This allows for easy construction of while loops. This essential for completing these two programs as they have a loop structure.

2. The initial design phase of the ISA was to build to the program 1 and 2 into MIPS language to optimize for least amount of registers and commands. From this process, we determined what commands were essential to complete the project and the best way to create loops needed for each program. The technique of loading and storing essential items for the program was determined to be the best. This allowed us to save registers and cut down on the complexity of the code. For program 1, we researched and found a memory saving technique that only required the multiplication of small numbers with repeated modding this allowed for a looping structure and easy implementation.

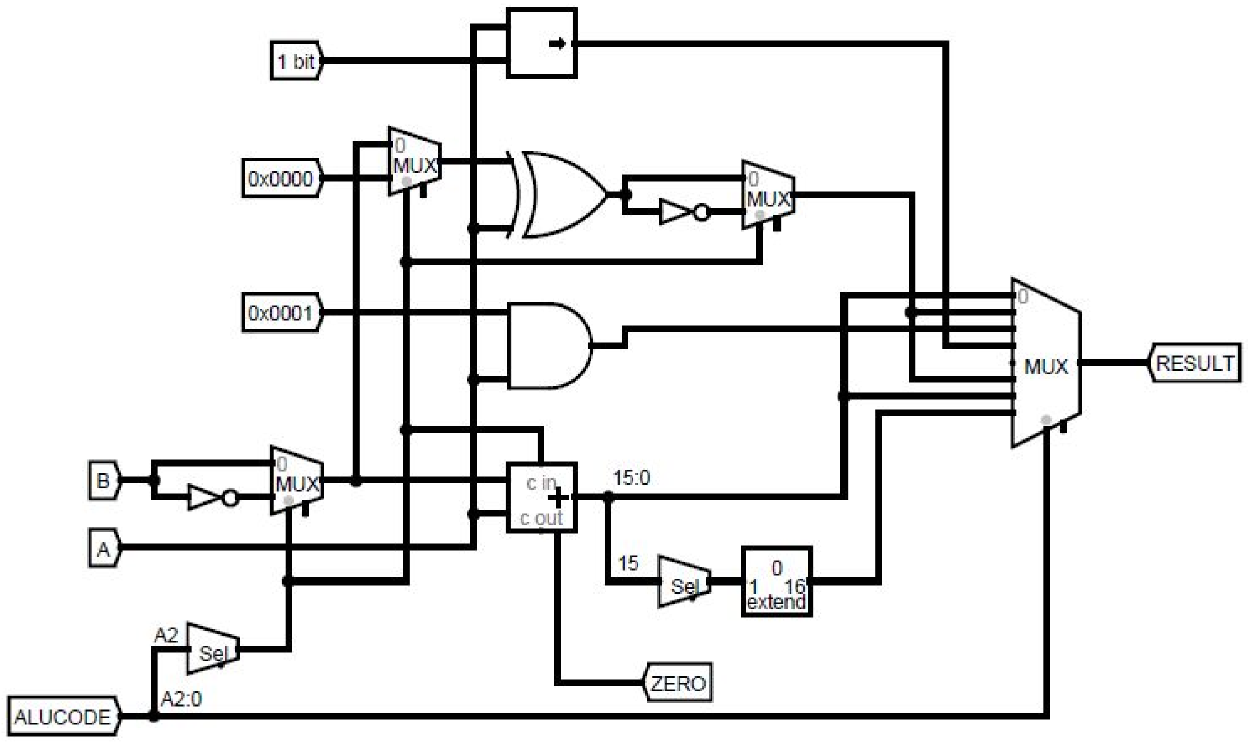
3. The changes that we would have made with an additional bit would be to either create a new instruction called add immediate value (addi) or recode the ISA by placing an extra 0 bit in front of our original instruction set and creating a jumpimm instruction that would then have a 7-bit number immediate. This would reduce the number of instruction used to load the register with the proper value before using the current jump instruction. If we had one less bit we would be forced to limit our register selections. For example, the load instruction would only be allowed to load memory into R0 and R1 instead of all four registers.

4. We divided the work up and helped each other design programs and modified each other's programs. Zach primarily focused on Python programming, Cameron focused primarily on program #2, and Teja focused primarily on program #1. The first objective was to decide a preliminary ISA design. This was done at our first meeting where we decided on which commands we thought were essential to the project. Then we decided to try to write both program 1 and program 2 in MIPS using as few as possible instructions and memory. Meanwhile, Zach began working on python code for an assembler and disassembler. By the first check, we had an ISA design and python code for an assembler and disassembler. The next week we began programming program 1 and 2 in our own ISA. Cameron primarily focused on program 2 during this week, while Zach and Teja focused on program 1. By the second check, we had completed program 2 and

made good progress on program 1. By the third week, we had completed both programs, Zach and Cameron began working on the hardware designs, and Teja began the report. Throughout the process, we continued to help each other on their tasks. Overall, we spent 30 hours collectively.

5. If given more time to optimize our design we would have looked into customing and reducing the hardware more although we attempted to reduce the hardware by combining the not and xor bitwise operation into a single gate in order for out jump command to work we had to add an addition adder which is a expensive piece of hardware compared to a mux. Another idea we considered was to use 3 bits to represent different combinations of our four registers. For example, 010 would be the combination that refers to R0, R1 as RX, RY respectively.

ALU Design

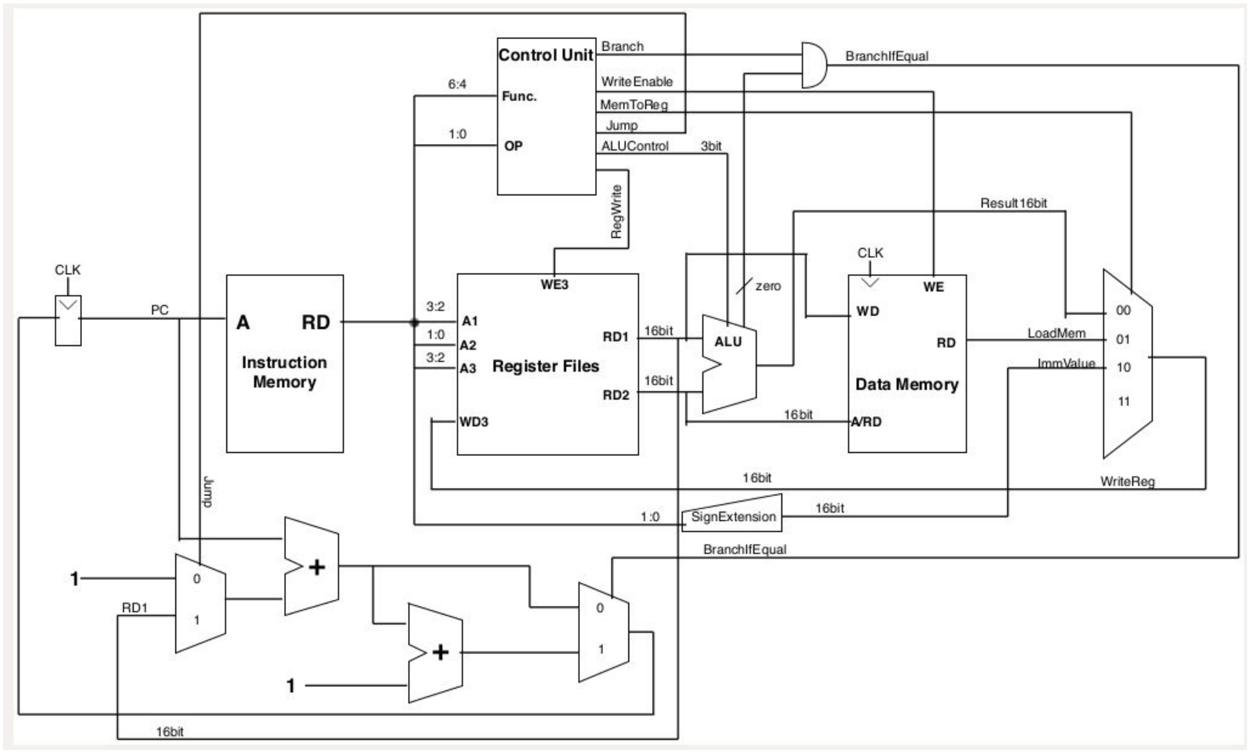


ALU DECODER TABLE:

NAME	FUNCTION	OP	ALUCODE
beq (Branch equal)	011	XX	110
slt (Set less than)	100	XX	111
not (Bitwise not)	110	11	101
add (Signed addition)	010	XX	000
xor (Bitwise xor)	111	01	001
slr (Logical shift right)	110	XX	011
and (Bitwise and with 1)	110	00	010

ISA CONTROL DECODE TABLE:

Instruction	Function	OP	RegWrite	Branch	Jump	WriteEnable	MemtoReg	ALU CONTROL 2:0
load	000	XX	1	0	0	0	01	XXX
store	001	XX	0	0	0	1	XX	XXX
beq	011	XX	0	1	0	0	XX	110
slt	100	XX	1	0	0	0	00	111
not	110	11	1	0	0	0	00	101
add	010	XX	1	0	0	0	00	000
and	110	01	1	0	0	0	00	010
xor	111	XX	1	0	0	0	00	001
slr	110	00	1	0	0	0	00	011
jump	110	10	0	0	1	0	XX	XXX
init	101	XX	1	0	0	0	10	XXX


Control Unit
Branch
WriteEnable
MemToReg
Jump
ALUControl
3bit
Func.
OP
6:4
1:0
RegWrite
BranchIfEqual
Result16bit
CLK
PC
A
RD
Instruction Memory
3:2
1:0
3:2
A1
A2
A3
WE3
Register Files
RD1
RD2
WD3
16bit
16bit
zero
ALU
16bit
CLK
WE
WD
RD
A/RD
Data Memory
LoadMem
ImmValue
00
01
10
11
WriteReg
16bit
1:0
SignExtension
16bit
Jump
BranchIfEqual
1
RD1
0
1
+
+
1
16bit