NORTHWESTERN
UNIVERSITY

Single Cycle Processor

EECS 361

Fall 2014

Group 6

Siddhartha Joshi
Shao-Han Chen
Panitan Wongse-ammat

Introduction:

In this project we have implemented a single-cycle processor. The processor handles the following subset of the MIPS instruction set:

• arithmetic: add, addi, addu, sub, subu • logical: and, or, sll
• data transfer: lw, sw
• conditional branch: beq, bne, slt, sltu

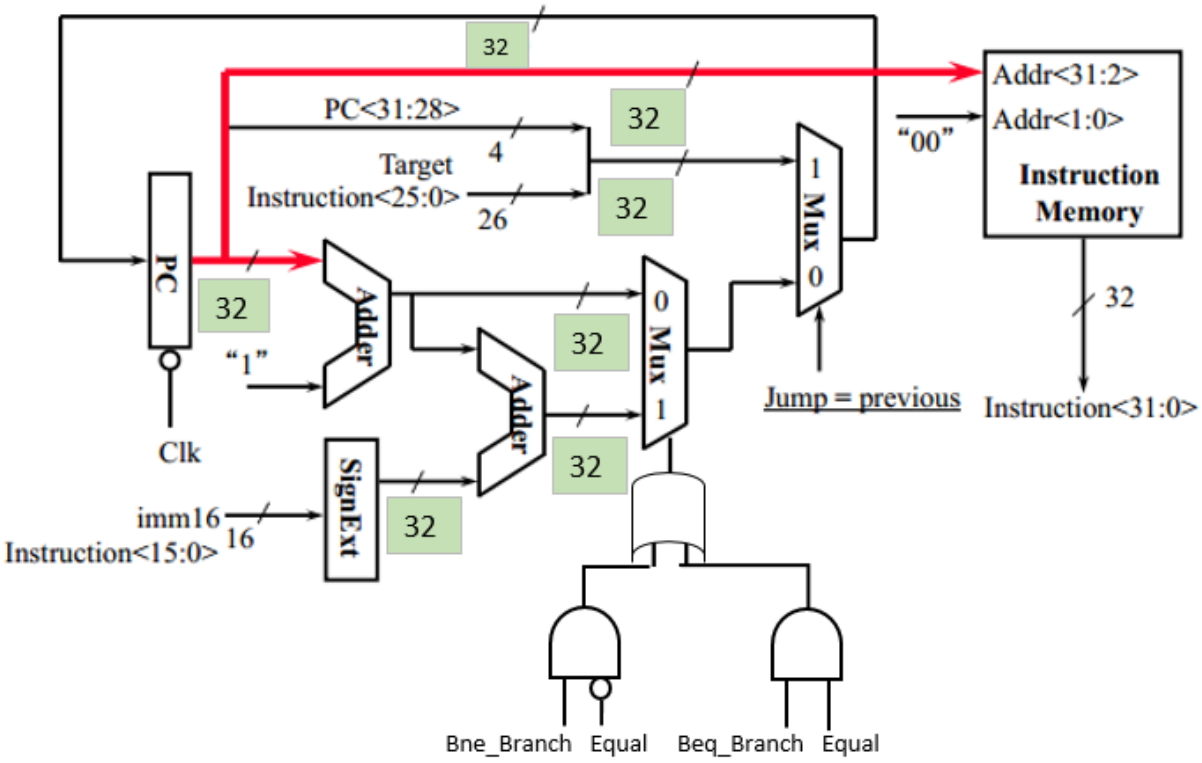
Our design focus was primarily on using maximum number of library elements which resulted in fast implementation. The biggest problem we faced was avoiding erroneous writes into memories (sram.vhd and syncsram.vhd) which almost all of the time. In the ModelSim Simulator, since we are doing functional simulations, all the components have zero delay which is not how we think about the design. Also if at a clock edge many inputs to a component change it results in glitches. Normally glitches were of no concern but when they occur at the input of memories they cause problems. We fixed this by using a clocked write into the register memory and a gated write at the input of the data memory.

All code is on github:

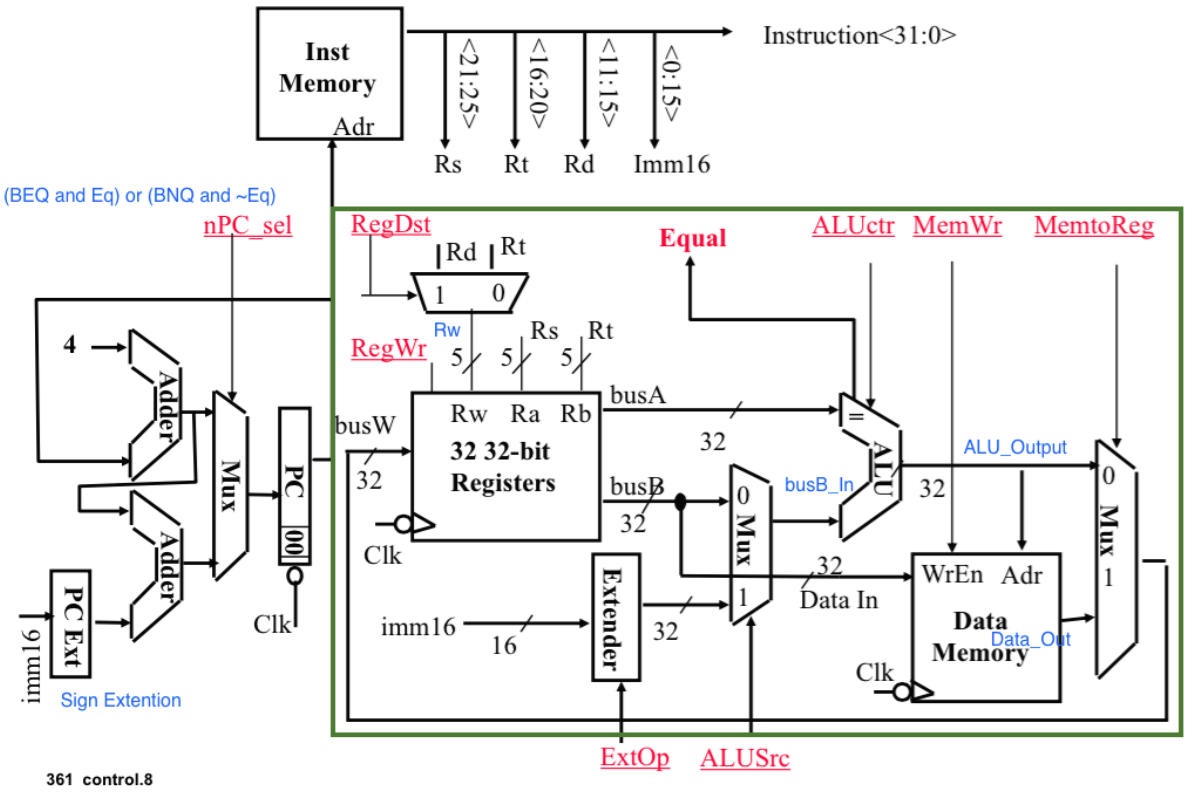
https://github.com/sidjos/Single_Cycle_Datapath

Instruction Fetch:

The most important issue in Instruction Fetch is to deal with branch condition. That is, if there's no branch, we simply fetch PC+4 for the next instruction. Otherwise, We should do PC+4+imm16*4.



Datapath:



Make 32 bit ALU using 1 bit ALUs. Bit slice design.

One bit ALU performs: ADD, SUB, AND and OR operations.
SLL (shift left logical), SLT(set on less than) and SLTU(set on less than unsigned) are implemented in an upper level of ALU).

Top level ALU is defined as:

```
entity alu is
  port(
     ctrl: in std_logic_vector (3 downto 0);
     A : in std_logic_vector(31 downto 0);
     B: in std_logic_vector ( 31 downto 0);
```

```
        cout: out std_logic;
        ovf: out std_logic;
        ze: out std_logic;
        R: out std_logic_vector(31 downto 0)
        );
end alu;
```

ALU control signals:

ALU	ALUCtr
Add	0000
Sub	0001
And	0010
Or	0011
Sll	0100
Slt	1001
Sltu	1101
Addi	0000
Addu	0000
Subu	0001
Lw	0000
Sw	0000
Beq	0001
Bne	0001

Control Logic:

Instruction type	Instr	Format	Format (binary)	Opcode (Hex)	Funct (Hex)
Arithmetic	add	R	0000 00ss ssst tttt dddd d000 00 10 0000	00	20
	addi	I	0010 00ss ssst tttt iiii iiii iiii iiii	08	-
	addu	R	0000 00ss ssst tttt dddd d000 00 10 0001	00	21
	sub	R	0000 00ss ssst tttt dddd d000 00 10 0010	00	22
	subu	R	0000 00ss ssst tttt dddd d000 00 10 0011	00	23
Logical	and	R	0000 00ss ssst tttt dddd d000 00 10 0100	00	24
	or	R	0000 00ss ssst tttt dddd d000 00 10 0101	00	25
	sll	R	0000 00ss ssst tttt dddd daaa aa 00 0000	00	00
Data	lw	I	'1000 11ss ssst tttt iiii iiii iiii iiii	23	-
Transfer	sw	I	'1010 11ss ssst tttt iiii iiii iiii iiii	2b	-
Conditional	beq	I	0001 00ss ssst tttt iiii iiii iiii iiii	4	-
Branch	bne	I	0001 01ss ssst tttt iiii iiii iiii iiii	5	-
Compare	slt	R	'0000 00ss ssst tttt dddd d000 0010 1010	00	2a
	sltu	R	'0000 00ss ssst tttt dddd d000 0010 1011	00	2b

Format	Meaning	RegDst	RegWr	ExtOp	ALUSrc	ALUctr	MemWr	MemtoR	Bne_brai
add $d, $s, $t	$d = $s + $t	1	1	x	0	Add	0	0	0
addi $t, $s, i	$t = $s + SignExt	0	1	1	1	Add	0	0	0
addu $d, $s, $t	$d = $s + $t	1	1	0	0	Add	0	0	0
sub $d, $s, $t	$d = $s - $t	1	1	x	0	sub	0	0	0
subu $d, $s, $t	$d = $s - $t	1	1	x	0	sub	0	0	0
and $d, $s, $t	$d = $s & $t	1	1	x	0	and	0	0	0
or $d, $s, $t	$d = $s | $t	1	1	x	0	or	0	0	0
sll $d, $t, i	$d = $s << shamt	1	1	0	1	sll	0	0	0
lw $t, i($s)	$t = MEM[R[rs]+SignExtImm]	0	1	1	1	add	0	1	0
sw $t, i($s)	MEM [R[rs]+SignExtImm] = $t	x	0	1	1	add	1	x	0
beq $s, $t, i	if ($s == $t) pc += i * 4 + 4	x	0	x	0	sub	0	x	0
bne $s, $t, i	if ($s != $t) pc += i * 4 + 4	x	0	x	0	sub	0	x	1
slt $d, $s, $t	$d = 1 if $s < $t ; 0 otherwise	1	1	x	0	slt	0	0	0
sltu $d, $s, $t	$d = 1 if $s < $t ; 0 otherwise	1	1	x	0	sltu	0	0	0

Issue and Challenges: For sll we know that it is the r-type instruction and the value in $t register will be shifted 5-bit shamt times. In the first ALU project that the shift value is the first 5 bits of the 32-bit second input of the ALU i.e bus B. We made some modification to send the shamt as 16-bit immediate to the alu through the extender. We determine the ALUSrc to be one so that the shift value will be zero extended to 32 bits and be selected by the multiplexer for the second input of the ALU. Other signals such as ALUCtr remain the same as the ALU project.

Note: x = don't care. However, we choose to set the don't care value to be zero.

Signal Description

RegDst = 1 select Rd ; 0 would be Rt otherwise
RegWr = 1 write to register file; 0 do nothing
ExtOp = 0 choose ZeroExtended value; =1 choose sign extended value
ALUSrc = 0 opt for BusB, 1 select Imm16
ALUCtr: as is on the ALU control table
MemWr = 1 write to memory ; 0 do nothing
MemtoReg = 1 :choose the data from ALU; = 0 the output data from data memory
Bne_branch = 1 for bne instruction
Beq_branch = 1 for beq instruction

References

Lectures slides of EECS 361 by Prof. Gokhan Memik, Computer Organization and Design, Fifth Edition: The Hardware/Software Interface by David A. Patterson and John L. Hennessy, TA sessions with Kaicheng and class lectures. Development was done on ModelSim PE Student Edition.