D0011E - Assignment 3

Peter Panduro

August 2018

1 Part 1 - Extend MIPS to support lw and sw

Load word (lw) and store word (sw), as the names suggests, loads and stores contents of specific addesses in a register. They are I-type instructions and containing one register address to the address where to load or store the information ($t), a base memory address ($s), and an offset. The syntaxes are
lw $t, offset($s)
sw $t, offset($s).

To make these instructions available to the design from assignment 2 a data memory is added together with another multiplexer to control what signal is sent to WD on the register file. (Fig. 1). The active signals for lw and sw paths are specified in table 1.

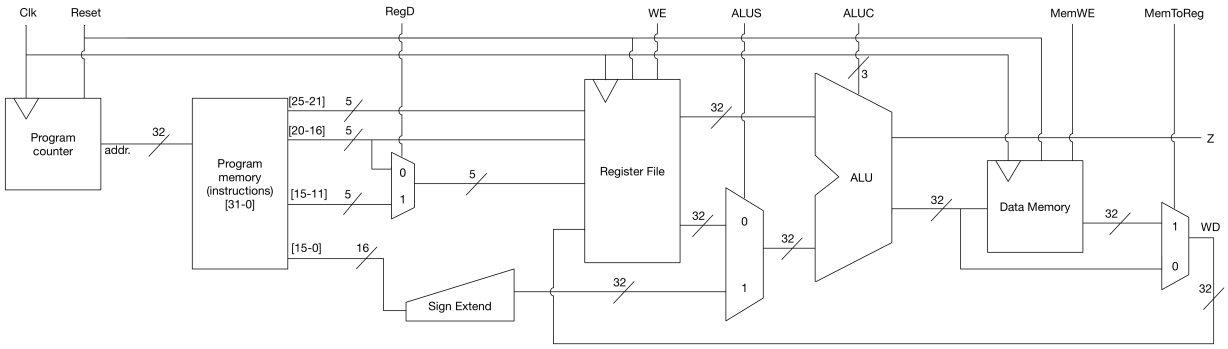

Figure 1: RTL schematic with support for lw and sw

Instruction	WE	ALUC	RegD	ALUS	MemWE	MemToReg
lw	1	010	0	1	0	1
sw	0	010	X	1	1	X

Table 1: Active signals of lw and sw

2 Part 2 - Extend MIPS to support beq

To make use of the commonly used if-statement a new instruction must be implemented. This instruction is called beq which stands for branch on equal. It is a I-type instruction and contains two register addresses which it compares and a 16-bit immediate value. This immediate value is a signed value and is used to calculate where to continue a program if the two register values are equal. This works by multiplying the immediate value by 4, because MIPS instruction addresses progresses in increments of 4, plus the next instruction address which is often referred as pc+4. The comparison of the registers are used by checking the Z-flag of the ALU and the multiplication of 4 is performed by 2 left shifts («2).

Because the main ALU is comparing the register addresses and outputs the result with the Z-flag it cannot be used for any other computation at the same time. The design can be found in figure 2 and the active signals is shown in table 2.

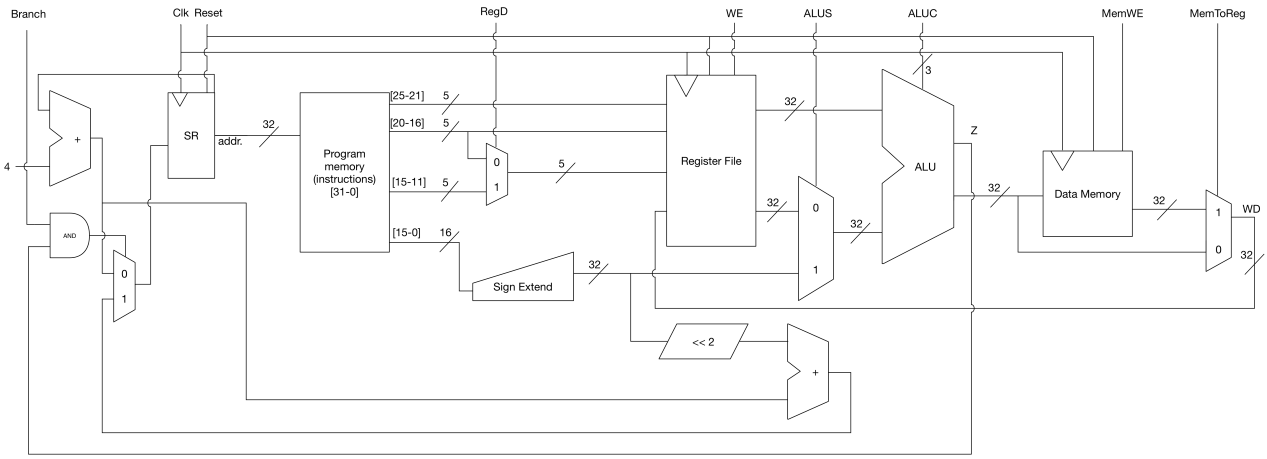

Figure 2: RTL schematic with support for beq

Instruction	WE	ALUC	RegD	ALUS	MemWE	MemToReg	Branch
beq	0	110	X	0	0	X	1

Table 2: Active signals of beq

3 Part 3 - Extend MIPS to support j

In order to make the processor more flexible and efficient a jump (j) instruction should be implemented. In contrast to beq which is a conditional jump to a

relative offset the j instruction lets the processor unconditionally jump to almost any address. The j instruction is its own type, a J-type instruction which only takes a 26-bit address.

Since the opcode is 6 bits long it is not possible to fit all 32-bit addresses. To address this issue the address is zero-extended to 32-bit and then left shifted by two bits. The address is then combined with the 4 most significant bits of the next address. This results in that the j instruction only will be able to jump to an address that shares the four most significal bits with the next address. The RTL schematic can be found at fig. 3 and table 3 lists the active signals for the j instruction.

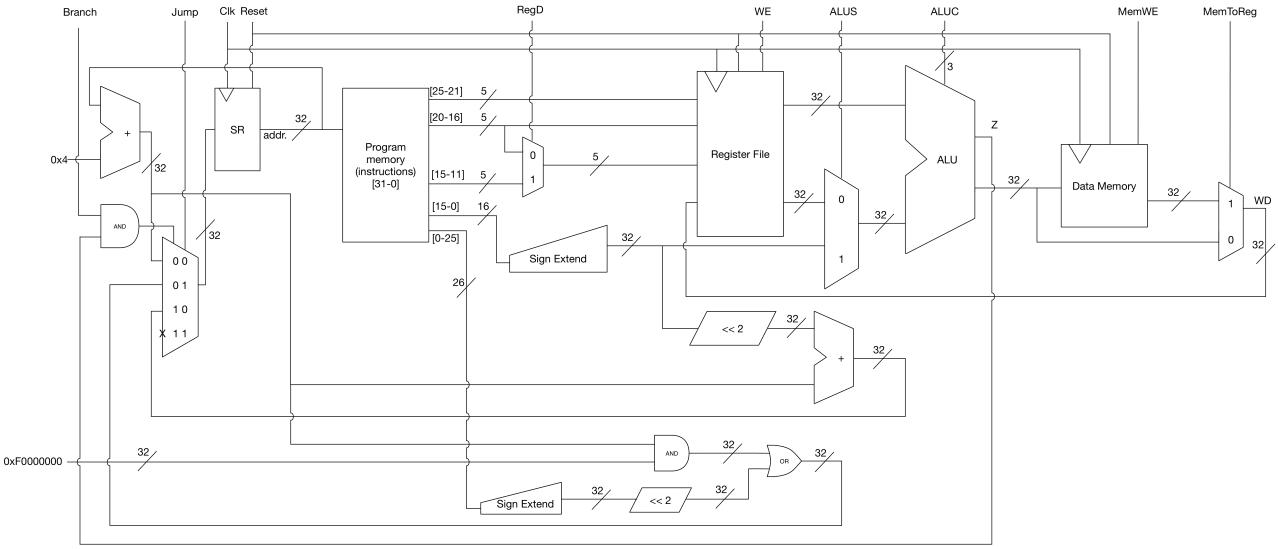

Figure 3: RTL schematic with support for j

Instruction	WE	ALUC	RegD	ALUS	MemWE	MemToReg	Branch	Jump
j	0	110	X	0	0	X	X	1

Table 3: Active signals of j

4 Part 4 - Truth table of instructions and control signals

Instr.	opcode	funct	WE	ALUC	RegD	ALUS	MemWE	MemToReg	Branch	Jump
ADD	0x0	0x20	1	010	1	0	0	0	0	0
ADDI	0x8	-	1	010	0	1	0	0	0	0
SUB	0x0	0x22	1	110	1	0	0	0	0	0
SLT	0x0	0x2A	1	111	1	0	0	0	0	0
SLTI	0xA	-	1	111	0	1	0	0	0	0
AND	0x0	0x24	1	000	1	0	0	0	0	0
OR	0x0	0x25	1	001	1	0	0	0	0	0
LW	0x23	-	1	010	0	1	0	1	0	0
SW	0x2B	-	0	010	X	1	1	X	0	0
BEQ	0x4	-	0	110	X	0	0	X	1	X
J	0x2	-	0	110	X	0	0	X	X	1

Table 4: Truth table of control signals and instructions

5 Part 5 - Assembly program

Using the supported instructions an assembly program that divides two positive integers N (Numerator) and D (Denominator) is written in machine code.
N (numerator) is stored in data memory location 0x0003.
D (denominator)is stored in data memory location 0x0004.
The result Q (quotient) shall be stored in location 0x0005.
The result R (remainder) shall be stored in location 0x0006.

The pseudo code for the algorithm is as follows:

```
Q:=0
R:=N
while R >= D do
    Q:=Q+1
    R:=R-D
end
return(Q,R)
end
```

As seen in table 5 the code is 10 instructions long. half of the instruction is setting upp the program by loading the value 0x1 in a register ($r1 in this case) to use later in the program and loading/saving the variables. The rest of the instructions are a loop that runs the same number of times as the denominator fits in the numerator, and the slt and beq instructions will run one more time during the last verification.

Assembly Code	Machine Code
addi $r1, $r1, 1	0010 0000 0010 0001 0000 0000 0000 0001
lw $r6, 0x3($r0)	1000 1100 0000 0110 0000 0000 0000 0011
lw $r4, 0x4($r0)	1000 1100 0000 0100 0000 0000 0000 0100
slt $r7, $r6, $r4	0000 0000 1100 0100 0011 1000 0010 1010
beq $r7, $r1, 0x3	0001 0000 1110 0001 0000 0000 0000 0011
addi $r5, $r5, 0x1	0010 0000 1010 0101 0000 0000 0000 0001
sub $r6, $r6, $r4	0000 0000 1100 0100 0011 0000 0010 0010
j 0x3	0000 1000 0000 0000 0000 0000 0000 0011
sw $r5, 0x5($r0)	1010 1100 0000 0101 0000 0000 0000 0101
sw $r6, 0x6($r0)	1010 1100 0000 0110 0000 0000 0000 0110

Table 5: Assembly and machine code of the division task