Jose Franco Baquera

April 16, 2019

Assignment 4

CS 473: Architectural Concepts I

1. We examine in detail how an instruction is executed in a single-cycle data path. Suppose the following instruction word is fetched to a single-cycle processor:

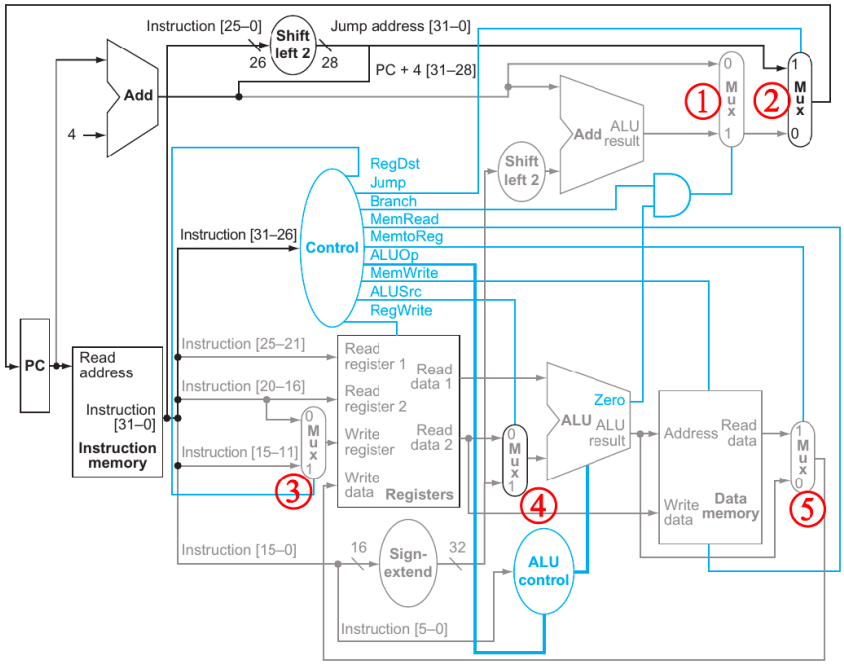
10101100011000100000000000010100

Assembly Equivalent = sw $v0, 20($v1) = sw r2, 20(r3)

Assume that data memory is all zeros and that the processors registers have the following values at the beginning of the cycle in which the above instruction word is fetched:

r0	r1	r2	r3	r4	r5	r6	r8	r12	r31
0	–1	2	–3	–4	10	6	8	2	–16

a. What are the outputs of the sign-extend and the jump Shift left 2 unit (near the top of the following figure) for this instruction word?



Sign Extended: The sign-extended component uses the immediate portion of the instruction, which, in MIPS, is the last 16 bits (i.e. Instruction[15-0]). To find the output of the sign-extended component, we first note that the last 16 bits of the given instruction are the following: 0b0000000000010100. As the datapath

demonstrates, we now sign-extend these 16 bits into 32 bits. That is, the final output is the following: 0b00000000000000000000000000010100

Answer: The output of the sign extended component is the following: 0b0000-0000-0000-0000-0000-0000-0001-0100. The equivalent hexadecimal number is 0x00000014.

Jump Shift Left 2: We use the jump shift left 2 component in the previous datapath when we have J-format instructions. That is, the jump shift left 2 component uses the immediate address portion of the instruction, which, in MIPS, is the last 26 bits. To find the output of the jump shift left 2 component, we first note that the last 26 bits of the given instruction are the following: 0b00011000100000000000010100. As the datapath demonstrates, we now shift this bit number to the left 2 position, which results in a 28-bit number: 0b00011000100000000000001010000.

Answer: The output of the jump shift left 2 component is the following: 0b0001-1000-1000-0000-0000-0101-0000. The equivalent hexadecimal number is 0x1880050.

b. What are the values of the ALU control unit's inputs for this instruction?

There are two input values for the ALU control unit: the 6-bit function field (i.e. the last 6 bits of the instruction) and the 2-bit ALUOp.

6-Bit Function Field

We note that the last 6 digits of the instruction are 0b010100.

Answer: The 6-bit function field input for the ALU control unit is the following: 0b010100.

2-Bit ALUOp

To get the 2-bit ALUOp input, we first use the fact that the control unit takes in the first 6 bits of the instruction (i.e. the instruction's opcode) and that one of its outputs is the 2-bit ALUOp. The first 6 bits of the instruction are the following: 0b101011 or, equivalent, 0x2B. Therefore, this is a store word instruction. According to the class PowerPoint, store word instructions will always have an ALUOp equal to 0b00. It is important to note that ALUOp1 = ALUOp0 = 0.

Answer: The 2-bit ALUOp input is the following: 0b00. (NOTE: ALUOp1 = ALUOp0 = 0.)

It is important to note that the ALU control unit's output will be discussed later on in homework assignment.

c. What is the new PC address after this instruction is executed?
We note that the instruction is a store word instruction. Therefore, we do not do any conditional branching or jumping. The PC address is therefore PC + 4.

Answer: The new PC address after this instruction is executed is the following: PC + 4.

d. For each MUX 1 through 5, show the values of its data output during the execution of this instruction and these register values.

MUX 1: We note that MUX 1 has three inputs, including PC + 4 (i.e. only taken if the branch is false), (PC + 4) + (Immediate x 4) (i.e. only taken if the branch is true; corresponds to the branch destination address), and the result of "AND" together with the zero output of the ALU and the control signal that indicates that the instruction is a branch.
Furthermore, we note the following:

PC + 4 => 0 MUX1
(PC + 4) + (Immediate x 4) => 1 MUX1

Since the result of this AND is zero (i.e. the branch control signal is zero, and anything we AND with zero is zero), then MUX 1 chooses 0 instead of 1. Therefore, the output of MUX 1 is PC + 4.

Answer: MUX 1 "chooses" 0 and has PC + 4 as output.

MUX 2: We note that MUX 2 also has three inputs, including the output of MUX 1, the jump address (i.e. PC + 4(31….28):Immediate x 4), and the control signal that indicates that the instruction is a jump.
Furthermore, we note the following:

MUX 1 Result = PC + 4 => 0 MUX2
Jump address => 1 MUX2

Since the result of the jump control signal is zero (i.e. the instruction is a store word), then MUX 2 chooses 0 instead of 1. Furthermore, the output of MUX 1 is PC + 4. Therefore, the output of MUX 2 is also PC + 4.

Answer: MUX 2 "chooses" 0 and has PC + 4 as output. The program counter register will also have this value after the instruction is finished executing.

MUX 3: We note that MUX 3 also has three inputs, including Instruction[20-16], Instruction[15-11], and the control signal that indicates that the instruction has a RegDst. Furthermore, we note the following:

Instruction[20-16] = 0b00010 => 0 MUX3

Instruction[15-11] = 0b00000 => 1 MUX3

Since the result of the RegDst control signal is "X", ***we do not care*** if MUX 3 chooses 0 or 1 (i.e. the save word instruction does not use the destination register). Therefore, the output MUX 3 is either 0b00010 = 0x2 or 0b00000 = 0x0.

Answer: MUX 3 "chooses" 0 or 1 and has either 0b00010 = 0x2 or 0b00000 = 0x0 as output. That is, the write register contains "X" since we do not care about writing into a register when dealing with sw instructions (RegDst is "X").

MUX 4: We note that MUX 4 also has three inputs, including read data 2, the 32-bit sign-extended Instruction[15-0], and the control signal that indicates that the instruction has an ALUSrc. Furthermore, we note the following:

Read Data 2 = $v0 = r2 = 2 => 0 MUX4

32-bit sign-extended Instruction[15-0] = 0x00000014 => 1 MUX4

Since the result of the ALUSrc control signal is 1, then MUX 4 chooses 1. Therefore, the output MUX 4 has the following output: 0b00000000000000000000000000010100.

Answer: MUX 4 "chooses" 1 and has 0b00000000000000000000000000010100 as the output. The equivalent hexadecimal number is 0x00000014, which is equal to 20 in decimal.

MUX 5: We note that MUX 5 also has three inputs, including read data from memory, ALU result, and the control signal that indicates that the instruction has an MEMtoReg. Furthermore, we note the following:

ALU Result = MUX 4 Result + Read Data 1

= 0x00000014 + $v1 = 0x00000014 + r3

= 0x00000014 + -3 = 0x11 =17=> 0 MUX5

Read Data from Memory = 0 (Directions states that we must "assume all data memory has zeros") => 1 MUX5

Since the result of the MEMtoReg is "X", then ***we do not care*** if MUX 5 chooses 0 or 1 (i.e. the save word instruction does not write to a register from memory).

Answer: MUX 5 "chooses" 0 or 1 and has either 0x11 (i.e. 17 in decimal) or 0x00 as output. That is, the output is "X" and the write data register will also contain "X" since we do not care about writing from memory into a register (MEMtoReg is "X").

e. For the ALU and the two add units, what are their data input values?

For the ALU unit, we note that there are three inputs: Read data 1, the output from MUX 4, and the output of the ALU control unit. Since we know that the equivalent assembly instruction is sw $v0, 20($v1) and that $v1 corresponds to r3, then the output of read data 1 is the contents of r3. Furthermore, we also know that the output of MUX 4 is 0x00000014 (i.e. 20 in decimal). The last input is the output of the ALU control unit, which is equal to 0b0010 = 0x2. Therefore, the inputs of the ALU unit is 0x00000014 (i.e. 20 in decimal), -3, and 0b0010 = 0x2.

Answer: ALU unit has data input values 0x00000014 = 20, -3 (decimal), and 0b0010 = 0x2.

For the first add unit (PC), we note that there are two inputs: The program counter and the constant 4 (i.e. in decimal form). Therefore, the first add unit has PC and 4 as input values.

Answer: The first add unit (PC) has data input values PC and 4.

For the second add unit (Branch Unit) we note that there are two inputs: The output of the first add unit and the sign-extended/shift left 2 binary number. The output of the first add unit is PC + 4. Furthermore, the sign extended/shift left 2 number is equal to 0b00000000000000000000000001010000 = 0x50 (or 80 in decimal). These are the two inputs.

Answer: The second add unit (Branch Unit) has data input values PC + 4 and 0x50 = 80.

f. What are the values of all inputs for the Registers unit?

There are a total of 5 inputs for the Register unit, including RegWrite, Instruction [25-21], Instruction [20-16], the output of MUX 3, and the output from MUX 5.

RegWrite: Since this is a save word instruction, we do not need to write to a register. This will be equal to 0.
Instruction [25-21]: 0b00011 = 0x3 = 3
Instruction [20-16]: 0b00010 = 0x2 = 2
Output of MUX 3: From part d, we found that the output of MUX 3 was “X” (i.e. 0x2 = 2 or 0x0 = 0).

Output of MUX 5: From part d, we found that the output of MUX 5 was "X" (i.e. 0x11 = 17 or 0x00 = 0).

Answer: The final answer is found on the following table. NOTE: All numbers are in decimal value.

Read Register 1 Input	Read Register 2 Input	Write Data Input	Write Register Input	RegWrite Input
3	2	(17 or 0) "Don't Care"	(2 or 0) "Don't Care"	0

2. Consider the following loop executed on a 5-stage pipeline.

```
loop:lw  r1,0(r1)
     and r1,r1,r2
     lw  r1,0(r1)
     lw  r1,0(r1)
     beq r1,r0,loop
```

Assume that perfect branch prediction is used (no stalls due to control hazards), and that the pipeline has full forwarding support. Also assume that many iterations of this loop are executed before the loop exits. Show a pipeline execution diagram for the third iteration of this loop, from the cycle in which we fetch the first instruction of that iteration up to (but not including) the cycle in which we can fetch the first instruction of the next iteration. Show all instructions that are in the pipeline during these cycles.

Instruction	**Stages**												
lw r1, 0(r1)	IF	ID	EX	MEM	WB								
and becomes NOP		bubble	bubble	bubble	bubble	bubble							
and r1, r1, r2			IF	ID	EX	MEM	WB						
lw r1, 0(r1)				IF	ID	EX	MEM	WB					
lw becomes NOP					bubble	bubble	bubble	bubble	bubble				
lw r1, 0(r1)						IF	ID	EX	MEM	WB			
beq becomes NOP							bubble	bubble	bubble	bubble	bubble		
beq becomes NOP								bubble	bubble	bubble	bubble	bubble	
beq r1, r0, loop									IF	ID	EX	MEM	WB

NOTE: Here is a more extended pipeline diagram that includes both the 2nd and 3rd iterations for question #2.

Ending of 2nd iteration.

Actual 3rd iteration.

Instruction	**Stages**												
lw r1, 0(r1)	WB												
beq becomes NOP	bubble	bubble											
beq becomes NOP	bubble	bubble	bubble										
beq r1, r0, loop	ID	EX	MEM	WB									
lw r1, 0(r1)	IF	ID	EX	MEM	WB								
and becomes NOP		bubble	bubble	bubble	bubble	bubble							
and r1, r1, r2			IF	ID	EX	MEM	WB						
lw r1, 0(r1)				IF	ID	EX	MEM	WB					
lw becomes NOP					bubble	bubble	bubble	bubble	bubble				
lw r1, 0(r1)						IF	ID	EX	MEM	WB			
beq becomes NOP							bubble	bubble	bubble	bubble	bubble		
beq becomes NOP								bubble	bubble	bubble	bubble	bubble	
beq r1, r0, loop									IF	ID	EX	MEM	WB

3. Assume that the following MIPS code is executed on a 5-stage pipeline, full forwarding, and a **predict-taken branch predictor**:

```
        lw r2,0(r1)
label1: beq r2,r0,label2 # not taken once, then taken
        lw r3,0(r2)
        beq r3,r0,label1 # taken
        add r1,r3,r1
label2: sw r1,0(r2)
```

Draw the pipeline execution diagram for this code, assuming that branches execute in the ID stage.

	Stages/Pipelining Cycles														
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw r2, 0(r1)	IF	ID	EX	MEM	WB										
beq becomes NOP		bubble	bubble	bubble	bubble	bubble									
beq becomes NOP			bubble	bubble	bubble	bubble	bubble								
beq r2, r0, label2				IF	ID	EX	MEM	WB							
sw r1, 0(r2) **(Flushed Instruction; Becomes NOP after instruction is fetched.)**					IF	bubble *(Instruction gets flushed here)*	bubble	bubble	bubble						
lw r3, 0(r2)						IF	ID	EX	MEM	WB					
beq becomes NOP							bubble	bubble	bubble	bubble	bubble				
beq becomes NOP								bubble	bubble	bubble	bubble	bubble			
beq r3, r0, label1									IF	ID	EX	MEM	WB		
beq r2, r0, label2										IF	ID	EX	MEM	WB	
sw r1, 0(r2)											IF	ID	EX	MEM	WB

4. The importance of having a good branch predictor depends on how often conditional branches are executed. Together with branch predictor accuracy, this will determine how much time is spent stalling due to mispredicted branches. In this exercise, assume that the breakdown of dynamic instructions into various instruction categories is as follows:

R-Type	BEQ	JMP	LW	SW
40%	25%	5%	25%	5%

Also, assume the following branch predictor accuracies:

Always-Taken	Always-Not-Taken	2-Bit
45%	55%	85%

a. Stall cycles due to mispredicted branches increase the CPI. What is the extra CPI due to mispredicted branches with the always-taken predictor? Assume that there are no data hazards, and that branch outcomes are determined in the EX stage, which means a misprediction causes 3 stalls.

We note the following information:

- 25% = .25 of the dynamic instructions are BEQ type.
- Always-taken branch predictor has a 45% = .45 accuracy. That is, we expect to have 100% - 45% = 55% = .55 chance of failure.
- Each mispredicted branch causes 3 stalls. In other words, there are 3 stalls per mispredicted branch instruction.
- Assume that stalls is a synonym for clock cycles. That is, one stall corresponds to one clock cycle.
- Total number of instructions = N.

Therefore, the extra CPI can be found by the following formula:

$$Proportion\ of\ BEQ\ Instructions * Chance\ of\ Failure * Cost\ of\ Failure * \frac{\cancel{N}}{\cancel{N}}$$

$$= .25 * (1 - .45) * 3 \frac{stalls}{one\ mispredicted\ branch\ instruction}$$

$$= 0.4125\ CPI$$

Answer: The extra CPI due to mispredicted branches with the always-taken predictor is 0.4125.

b. Repeat (a) for the always-not-taken predictor.

We note the following information:

- 25% = .25 of the dynamic instructions are BEQ type.

- Always-not-taken branch predictor has a 55% = .55 accuracy. That is, we expect to have 100% - 55% = 45% = .45 chance of failure.
- Each mispredicted branch causes 3 stalls. In other words, there are 3 stalls per mispredicted branch instruction.
- Assume that stalls is a synonym for clock cycles. That is, one stall corresponds to one clock cycle.
- Total number of instructions = N.

Therefore, the extra CPI can be found by the following formula:

$$Proportion\ of\ BEQ\ Instructions * Chance\ of\ Failure * Cost\ of\ Failure * \frac{\cancel{N}}{\cancel{N}}$$

$$= .25 * (1 - .55) * 3 \frac{stalls}{one\ mispredicted\ branch\ instruction}$$

$$= 0.3375\ CPI$$

Answer: The extra CPI due to mispredicted branches with the always-not-taken predictor is 0.3375.

c. Repeat (a) for the 2-bit predictor.

We note the following information:

- 25% = .25 of the dynamic instructions are BEQ type.
- 2-bit branch predictor has an 85% = .85 accuracy. That is, we expect to have 100% - 85% = 15% = .15 chance of failure.
- Each mispredicted branch causes 3 stalls. In other words, there are 3 stalls per mispredicted branch instruction.
- Assume that stalls is a synonym for clock cycles. That is, one stall corresponds to one clock cycle.
- Total number of instructions = N.

Therefore, the extra CPI can be found by the following formula:

$$Proportion\ of\ BEQ\ Instructions * Chance\ of\ Failure * Cost\ of\ Failure * \frac{\cancel{N}}{\cancel{N}}$$

$$= .25 * (1 - .85) * 3 \frac{stalls}{one\ mispredicted\ branch\ instruction}$$

$$= 0.1125\ CPI$$

Answer: The extra CPI due to mispredicted branches with the 2-bit predictor is 0.1125.