CS 151B: Homework 2

Jonathan Woong

804205763

Spring 2017

Discussion 1A

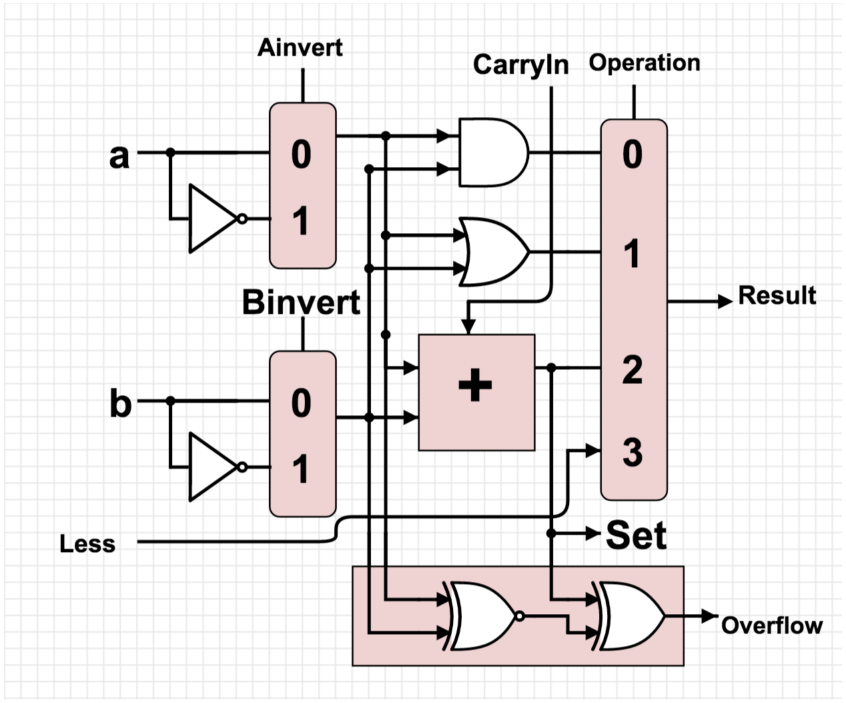
Monday 17th April, 2017

Problem 1

The ALU supported set on less than (`slt`) using just the sign bit of the adder. Let's try a set on less than operation using the values -7_{10} and 6_{10}. To make it simpler to follow the example, let's limit the binary representations to 4 bits: 1001_2 and 0110_2.

$$1001_2 - 0110_2 = 1001_2 + 1010_2 = 0011_2$$

This result would suggets that $-7 > 6$, which is clearly wrong. Hence, we must factor in overflow in the decision. Modify the 1-bit ALU in Figure B.5.10 on page B-33 to handle `slt` correctly.



The diagram shows the 1-bit ALU for the most significant bit. As described in the book, the Set signal is wired to the Less port of the least significant bit. The overflow detector works by XNOR-ing a and b, producing a signal of 1 when both a and b are 0 or when both a and b are 1. This is XOR-ed with the set bit, which is 0 when $a + b = 0$ and 1 when $a + b = 1$. Thus, if a and b are both 0 and the set bit is 1, there must be an overflow (the Overflow signal will be 1). The same argument can be used for when a and b are both 1 and the set bit is 0.

Problem 2

A simple check for overflow during addition is to see if the CarryIn to the most significant bit is not the same as the CarryOut of the most significant bit. Prove that this check is the same as in Figure 3.2.

Case	Operation	Operand A	Operand B	Result Indicating Overflow
1	$A+B$	≥ 0	≥ 0	< 0
2	$A+B$	< 0	< 0	≥ 0

Let n be the index of the most significant bit.

Case 1: Since A and B are both ≥ 0, A_n and B_n are both 0.
If CarryIn for n is 0, CarryOut for n can never be 1.
If CarryIn for n is 1, CarryOut must be 0. $A_n + B_n +$ CarryIn $= 0 + 0 + 1$, so the most significant bit of the result must be 1, and the result is a negative number. This is consistent with overflow.

Case 2: Since A and B are both < 0, A_n and B_n are both 1.
If CarryIn for n is 0, CarryOut for n must be 1. $A_n + B_n +$ CarryIn $= 1 + 1 + 0 = 0$, so the most significant bit of the result must be 0, and the result is a positive number. This is consisten with overflow.
If CarryIn for n is 1, CarryOut can never be 1.

Problem 3

The single cycle implementation of the MIPS processor uses for the main ALU the implementation shown in Figures B.5.10-B.5.12 (pp. B-33 - B-36) in the book. Due to a circuit malfunction, one of the four ALU control lines is always one (stuck-at-1). Specifically, refer to Figure B.5.13, number the control bits in little endian order, with the least-significant bit being bit-0. The faulty bit described above is bit-1.
The rest of the circuitry of the ALU and the rest of the processor operates normally.
Explain in full detail what will be the consequences of this fault when the processor executes programs - how will it change the behavior of the processor as observed by a user/programmer who does not know and does not care how the processor is implemented internally? Be sure to clearly identify each and every consequence of this fault.

Bit	**Label**
0	Ainvert
1	Binvert
2,3	Operation

If bit-1 is always 1, then all operations will be on the inverse of B rather than B. The table below shows the ALU control lines, function, and logical result.

Case	**ALU control lines**	**Function**	**Result**
1	0100	AND	$A \wedge \neg B$
2	0101	OR	$A \vee \neg B$
3	0110	add	$A + \neg B$
4	0110	subtract	unchanged
5	0111	set on less than	unchanged
6	1100	NOR	unchanged

Circuit level

Case 1: $A \wedge \neg B$

A	B	$\neg B$	$A \wedge \neg B$
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

Case 2: $A \vee \neg B$

A	B	$\neg B$	$A \vee \neg B$
0	0	1	1
0	1	0	0
1	0	1	1
1	1	0	1

Case 3: $A + \neg B$

A	B	$\neg B$	$A + \neg B$
0	0	1	1
0	1	0	0
1	0	1	0 (CarryOut=1)
1	1	0	1

Programmer level

Case 1: AND

The programmer will experience incorrect results when performing logical AND on two operands. The only case in which the result is 1 (or True) is when A is True and B is False. All other cases will result in False.

Case 2: OR

The programmer will experience incorrect results when performing logical OR on two operands. The result is False when A is false and B is True. All other cases will result in True.

Case 3: add

The programmer will experience subtraction rather than addition.

Problem 4

The fault to test for is whether the "MemRead" control signal becomes 0 if the RegDst control signal is 0, no fault otherwise.

When RegDst is 0, the register destination number for the Write register comes from the rt field. If MemRead is stuck-at-0, data from memory will be read for every instruction. For instructions that do not load from memory, the value read from memory will be discarded by the Mux that selects the value to write to the Register unit. However, since we cannot design an opcode that isolates the fault (opcode is zero and MemRead is 1), we cannot test for this fault.

Problem 5

The fault to test for is whether the "Jump" control signal becomes 0 if RegDst control signal is 0, no fault otherwise.

If Jump is stuck-at-1, the PC will always update as if the previous instruction was a Jump. We can test for this by executing an ADDI instruction with a non-zero immediate operand. We expect to see that the PC will not be pointing to the instruction directly following ADDI if Jump is stuck-at-1. However, since the opcode of the Jump instruction is non-zero, we cannot design an opcode isolates the fault (opcode is zero and Jump is 1).

Problem 6

Consider the single cycle implementation shown in Figure 4.24 (page 271). Assume that during one particular cycle, regardless of the inputs to the control unit, the control unit generates the following values for the control signals:

```
0001000001
```

The values of the control signals are specified in order from top to bottom, as they appear in the figure: the left-most bit is the value of the RegDst signal and the right-most bit is the value of the RegWrite signal. The most-significant bit of ALUOp appears to the left of the least-significant bit of ALUOp.

Specify all the state changes that will occur over this one cycle. Be sure to indicate which (if any) state elements will change their state and to what will be the new value of each of those state elements. Your answer must be as specific as possible. Explain your answer.

Signal	**Value**	**Effect**
RegDst	0	The register destination number for the Write register comes from the rt field
Jump	0	None
Branch	0	None
MemRead	1	Data memory contents designated by the address input are put on the Read data output
MemtoReg	0	The value fed to the register Write data input comes from the ALU
ALUOp1	0	Load/Store word
ALUOp0	0	Load/Store word
MemWrite	0	None
ALUSrc	0	The second ALU operand comes from the second register file output (Read data 2)
RegWrite	1	The register on the Write register input is written with the value on the Write data output

State changes

1. PC ← PC + 4: determined by Jump and Branch, and occurs for every clock cycle
2. Write register ← $rt: determined by RegDst
3. ALU operand 1 ← $rs: default behavior

ALU operand 2 ← $rt: determined by ALUSrc

Address ← $rs + $rt: determined by ALUOp

4. Write data ← Address: determined by MemtoReg
5. Read data ← M[Address]: determined by MemRead and MemWrite
6. $rt ← Address: determined by RegWrite

Problem 7

Add a new instruction to the single-cycle implementation described in Chapter 4. Note that this new instruction is not part of the standard MIPS ISA. The new instruction is `lwrr` (load word register-register). This is an R-format instruction that has similar functionality to the standard `lw` instruction. The difference is that the address is the sum of the value in the $rs register and the $rt register. The value is loaded from memory to the register specified in the $rd field.

Your starting point is the datapath shown in Figure 4.17 on page 265 and the control shown in Figure 4.18 on page 266.

Key building blocks of the datapath are implemented as shown in Appendix B: the ALU in Section B.5 and the register file in Section B.8.

All the instructions supported by the design in Figure 4.17 and 4.18 must continue to work correctly. You should minimize changes to the datapath and to the control. You need to show all the required changes to Figure 4.17 and Figure 4.18. If any modifications to the datapath building blocks (ALU, register file, etc) are necessary, you must show the implementation of the modified building blocks in detail.

The control line signals for `lwrr` are shown below:

Instruction	RegDst	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp
`lwrr`	1	1	1	1	0	0	10

The instruction for `lwrr` is shown below with the opcode chosen arbitrarily (must be currently unused):

opcode	rs	rt	rd	shamt	funct
001100	rs	rt	rd	00000	100000

No modifications need to be made to the existing logical units, as all of the necessary operations are already implemented. All that needs to be done is to modify the Control unit to decode our arbitrarily chosen opcode into the correct control signals:

RegDst: specifies the destination register $rd

MemToReg: specifies that data written to $rd comes from memory

RegWrite: specifies that Write data must be written to $rd

MemRead: specifies that Read data comes from M[$rs + $rt]

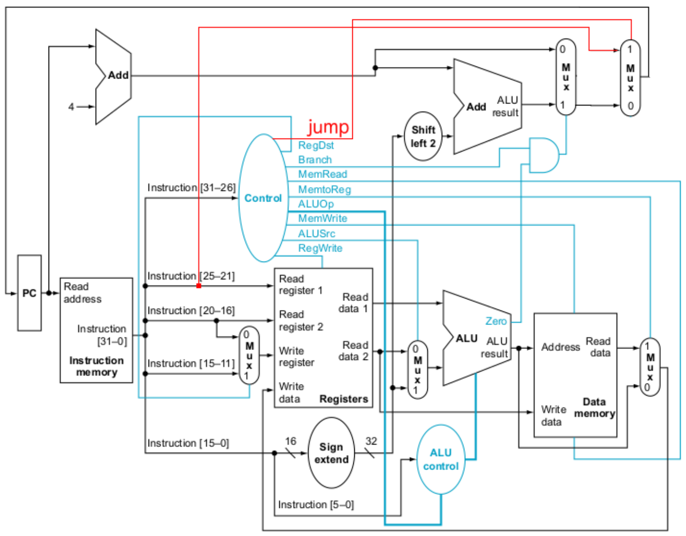
ALUOp and **funct**: specifies addition for the ALU

Problem 8

Consider the single cycle MIPS implementation with the datapath shown in Figure 4.17 (page 265) in the book and the control logic implemented based on the truth table in Figure 4.22 (page 269).

Modify this implementation to add support for the `jr` (jump register) instruction, which is part of the MIPS ISA. This instruction is described on page A-64 of the book.

A) Show any required modifications to the datapath. You can show the modifications on a copy of Figure 4.17. Also explain the modifications in words, to make sure your modifications are clear. In addition, if you need to modify any of the datapath building blocks, draw the implementation of the modified building blocks and explain the modifications in full detail.



A path is added that connects $rs to a multiplexer. A control signal labeled "jump" is added in order to control the output of the multiplexer. If the "jump" signal is 1, the multiplexer will output the value in $rs and the PC will become the value in $rs. If the "jump" signal is 0, the MIPS implementation operates as if the additions were never made.

B) Are there any new control signals required? If so, list them with an explanation and identify them on the datapath diagram.

Same as in part A.

C) Show any necessary changes to the control logic by presenting a modified version of the control truth table in Figure 4.22. For any entries that you add, mark all "don't cares" with an **X**.

We choose an arbitrary unused opcode as input to the Control unit.

The truth table is shown below:

Instruction	Jump	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
`jr`	1	X	X	X	0	0	0	0	X	X