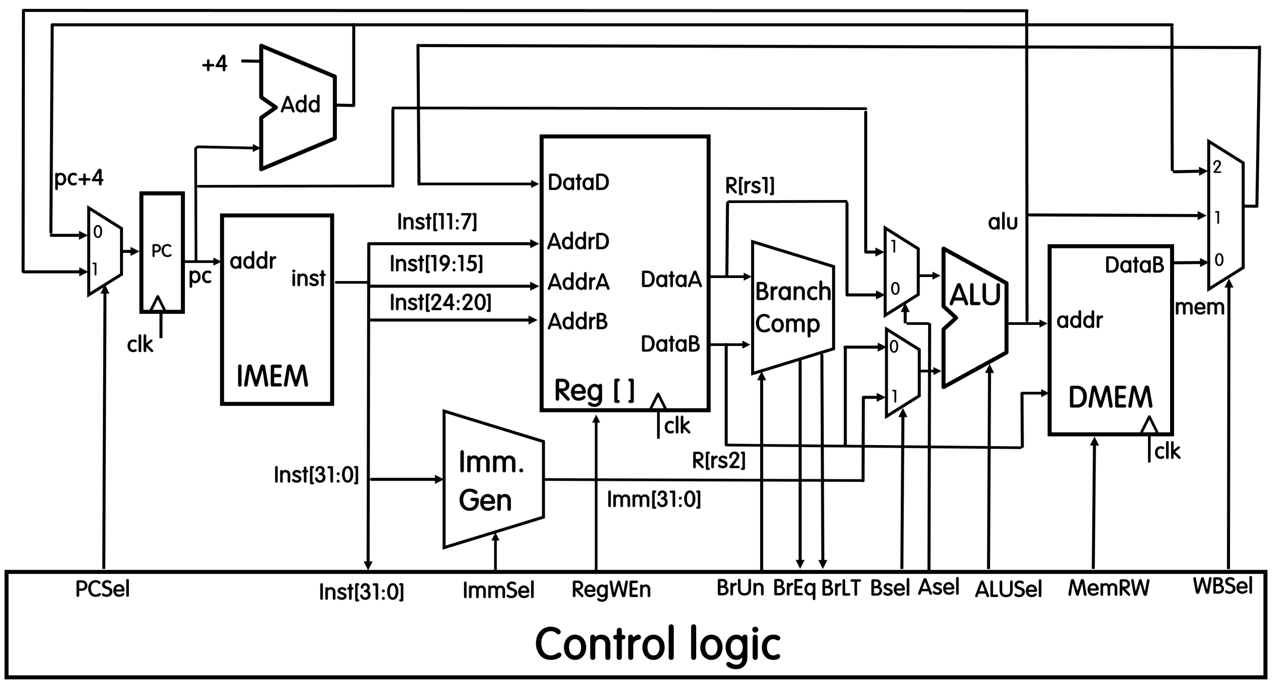
HW6.4. Datapath control signals

Consider the following datapath:



Suppose we have the following options for the ALUSel and ImmSel:

ALUSel	ALU operation
000	ADD: Result = A + B
001	SUB: Result = A - B
010	AND: Result = A & B
011	OR: Result = A | B
100	XOR: Result = A ^ B
101	SLL: Result = A << B
110	SRL: Result = A >> B
111	Pass B: Result = B

ImmSel	Immediate format
00	I-type
01	S-type
10	B-type
11	J-type

Now we are tasked to write the control signals for different instructions.

For each instruction listed below, write the corresponding control signals as a group of 13-bit binary code that follows the bit order below:

PCSel	RegWEn	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	WBSel
1 bit	1 bit	2 bits	1 bit	1 bit	1 bit	3 bits	1 bit	2 bits

<u>For control signals that are not relevant to the instruction, put "X" for every bit of the control signal</u>

As an example, the `addi` instruction will have the following control signals:

PCSel	RegWEn	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	WBSel
0	1	00	X	0	1	000	0	01

Here's the explanation for the `addi` control signals:

PCSel = 0, because you just execute the next instruction after `addi` if ever (it's not a jump or branch instruction), so you just select pc+4 branch. PC+4 branch corresponds to multiplexer input 0.

RegWEn = 1, because we have to write back to the register file to save the result of the `addi` instruction to rd.

ImmSel = 00, because `addi` is an I-type instruction, so you take in the I-type immediate. This corresponds to 00 according to the provided table.

BrUn = X, we technically don't care about this signal because we are not executing a branch instruction, so we put "X".

ASel = 0, because we have to select the output from the register file (this is rs1). Register file output (rs1) corresponds to the multiplexer input 0.

BSel = 1, because we have to select the immediate for the `addi` instruction. Immediate generator output corresponds to the multiplexer input 1.

ALUSel = 000, because we have to select the addition operation on the ALU. This corresponds to 000 according to the provided table.

MemRW = 0, because we don't have to write to the memory.

WBSel = 01, because we have to select the output of the ALU to be written back to the register file. ALU output corresponds to the multiplexer input 1 (so binary 01).

This 13-bit code can then be written as `0100X01000001`

Now it's your turn. For the following instructions, write the corresponding 13-bit binary string (without the 0b prefix) that would correspond to the control signals to properly execute the instruction. **All answers should be 13 bits long.**

`or` instruction

01XXX00011001 ✓ 100%

`lw` instruction

0100X01000000 ✓ 100%

`sw` instruction

0001X010001XX ✓ 100%

`jal` instruction

1111X11000010 ✓ 100%

`bgeu` instruction (*assume that the branch condition is true in this case*)

10101110000XX ✓ 100%

Try a new variant

Correct answer

`or` instruction

`01XXX00011001`

PCSel = 0, because you just execute the next instruction after `or` if ever (it's not a jump or branch instruction), so you just select pc+4 branch. PC+4 branch corresponds to multiplexer input 0.

RegWEn = 1, because we have to write back to the register file to save the result of the `or` instruction to rd.

ImmSel = XX, because `or` is an R-type instruction, we don't care about the immediate field at all, so we put "XX" for the 2 bits of ImmSel.

BrUn = X, we technically don't care about this signal because we are not executing a branch instruction, so we put "X"

ASel = 0, because we have to select the output from the register file (this is rs1). Register file output (rs1) corresponds to the multiplexer input 0.

BSel = 0, because we have to select the output from the register file (this is rs2). Register file output (rs2) corresponds to the multiplexer input 0.

ALUSel = 011, because we have to select the OR operation on the ALU. This corresponds to 011 according to the provided table.

MemRW = 0, because we don't have to write to the memory.

WBSel = 01, because we have to select the output of the ALU to be written back to the register file. ALU output corresponds to the multiplexer input 1 (so binary 01).

This 13-bit code can then be written as `01XXX00011001`

`lw` instruction

`0100X01000000`

PCSel = 0, because you just execute the next instruction after `lw` if ever (it's not a jump or branch instruction), so you just select pc+4 branch. PC+4 branch corresponds to multiplexer input 0.

RegWEn = 1, because we have to write back to the register file to save the result of the `lw` instruction to rd.

ImmSel = 00, because `lw` is an I-type instruction, so you take in the I-type immediate. This corresponds to 00 according to the provided table.

BrUn = X, we technically don't care about this signal because we are not executing a branch instruction, so we put "X"

ASel = 0, because we have to select the output from the register file (this is rs1). Register file output (rs1) corresponds to the multiplexer input 0.

BSel = 1, because we have to select the immediate for the `lw` instruction. Immediate generator output corresponds to the multiplexer input 1.

ALUSel = 000, because we have to select the addition operation on the ALU to do rs1 + offset for the memory address. This corresponds to 000 according to the provided table.

MemRW = 0, because we don't have to write to the memory.

WBSel = 00, because we have to select the output of the memory to be written back to the register file. Memory output corresponds to the multiplexer input 0 (so binary 00).

This 13-bit code can then be written as `0100X01000000`

`sw` instruction

```
0001X010001XX
```

PCSel = 0, because you just execute the next instruction after `sw` if ever (it's not a jump or branch instruction), so you just select pc+4 branch. PC+4 branch corresponds to multiplexer input 0.

RegWEn = 0, because `sw` instruction only writes to the memory, not the register file, so this should be 0.

ImmSel = 01, because `sw` is an S-type instruction, so you take in the S-type immediate. This corresponds to 01 according to the provided table.

BrUn = X, we technically don't care about this signal because we are not executing a branch instruction, so we put "X"

ASel = 0, because we have to select the output from the register file (this is rs1). Register file output (rs1) corresponds to the multiplexer input 0.

BSel = 1, because we have to select the immediate for the `sw` instruction. Immediate generator output corresponds to the multiplexer input 1.

ALUSel = 000, because we have to select the addition operation on the ALU to do rs1 + offset for the memory address. This corresponds to 000 according to the provided table.

MemRW = 1, because `sw` instruction writes to the memory, so this should be 1.

WBSel = XX, because `sw` instruction does not write to the register file (RegWEn = 0 anyway), so we don't care what WBSel is. Thus, we put XX for this control signal.

This 13-bit code can then be written as `0001X010001XX`

`jal` instruction

```
1111X11000010
```

PCSel = 1, because this is a jump instruction, we select the ALU branch to calculate PC + offset. ALU branch corresponds to multiplexer input 1.

RegWEn = 1, because `jal` instruction needs to update the register file to contain the return address, so this should be 1.

ImmSel = 11, because `jal` is a J-type instruction, so you take in the J-type immediate. This corresponds to 11 according to the provided table.

BrUn = X, we technically don't care about this signal because we are not executing a branch instruction, so we put "X"

ASel = 1, because we have to select the PC to calculate PC + offset for the jump target address. PC corresponds to the multiplexer input 1.

BSel = 1, because we have to select the immediate for the `jal` instruction. Immediate generator output corresponds to the multiplexer input 1.

ALUSel = 000, because we have to select the addition operation on the ALU to do PC + offset for the jump target address. This corresponds to 000 according to the provided table.

MemRW = 0, because we don't have to write to the memory.

WBSel = 10, because `jal` instruction needs to write the return address (PC + 4) to the register file. This corresponds to the multiplexer input 2 (so binary 10).

This 13-bit code can then be written as `1111X11000010`

`bgeu` instruction (*assume that the branch condition is true in this case*)

`10101110000XX`

PCSel = 1, because this is a branch instruction and the branch condition is true (so we will jump), we select the ALU branch to calculate PC + offset. ALU branch corresponds to multiplexer input 1.

RegWEn = 0, because `bgeu` instruction does not write to the register file. Thus, this should be 0.

ImmSel = 10, because `bgeu` is a B-type instruction, so you take in the B-type immediate. This corresponds to 10 according to the provided table.

BrUn = 1, because `bgeu` is an unsigned branch instruction, so this should be 1. If the branch is signed, such as `bge`, then this will be 0.

ASel = 1, because we have to select the PC to calculate PC + offset for the branch target address. PC corresponds to the multiplexer input 1.

BSel = 1, because we have to select the immediate for the `bgeu` instruction. Immediate generator output corresponds to the multiplexer input 1.

ALUSel = 000, because we have to select the addition operation on the ALU to do PC + offset for the branch target address. This corresponds to 000 according to the provided table.

MemRW = 0, because we don't have to write to the memory.

WBSel = XX, because `bgeu` instruction does not write to the register file (RegWEn = 0 anyway), so we don't care what WBSel is. Thus, we put XX for this control signal

This 13-bit code can then be written as `10101110000XX`

Submitted answer 2 correct: 100%

Submitted at 2022-10-23 08:28:42 (PDT)

hide

`or` instruction

`01XXX00011001` ✓100%

`lw` instruction

`0100X01000000` ✓100%

`sw` instruction

`0001X010001XX` ✓100%

`jal` instruction

`1111X11000010` ✓100%

`bgeu` instruction (*assume that the branch condition is true in this case*)

`10101110000XX` ✓100%

Submitted answer 1 **partially correct: 60%**

Submitted at 2022-10-23 08:28:16 (PDT)

show