Video explanation of solution is provided below the problem.

For all Beta related questions, you should make use of the Beta documentation, the Beta Instruction Summary, and the Beta Diagram.

A Better Beta

47/48 points (ungraded)
Marketing has decided that the next model Beta needs several additional instructions, and has called you in as a consultant to decide, in each case, whether

- **Macro**: the instruction can be implemented simply as a macro, whose body contains a single existing Beta instruction that performs the indicated operation
- **CTL**: he instruction can be implemented using the existing data paths, a new opcode and appropriate control signal generation to the Beta's control ROM
- **Hardware**: the instruction cannot be implemented without changes to the Beta's data paths.

For each of the following proposed new instructions, you are to determine whether it can be translated (using a macro) to a single existing instruction, and, if so, to write the equivalent assembly language instruction, **otherwise write NONE for the assembly instruction**. If it can't be translated to an existing instruction, you must determine whether it can be implemented as a new opcode using existing Beta data paths (including your ALU from the lab), and, if so, to specify appropriate control signals for that opcode. **If the implementation strategy is either Macro or Hardware, select NONE for each control signal value.**

1. An instruction that swaps the contents of two registers, in a single clock cycle:

```
SWAPR(Rx, Ry)   // Swap register contents
  TMP ← Reg[Rx]
  Reg[Rx] ← Reg[Ry]
  Reg[Ry] ← TMP
  PC ← PC + 4
```

Best implementation strategy

- ○ Macro
- ○ CTL
- ◉ Hardware

✔

If best implementation is Macro, enter the equivalent instruction, otherwise enter NONE. Do not include any white space in your answer.

NONE ✔ Answer: NONE

If best implementation is CTL, select the appropriate value for each control signal, otherwise select NONE for each control signal.

Instr	ALUFN	WERF	BSEL	WDSEL	MOE	MWR	RA2SEL	PCSEL	ASEL	WASEL
SWAPR	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE
	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE

Explanation
There is no way to write to two different registers within a single clock cycle using the Beta's current hardware.

2. An instruction that negates the two's-complement integer in Rx:

```
NEG(Rx, Ry)   // two's complement negate
  Reg[Ry] ← – Reg[Rx]
  PC ← PC + 4
```

Best implementation strategy

- ◉ Macro
- ○ CTL
- ○ Hardware

✔

If best implementation is Macro, enter the equivalent instruction, otherwise enter NONE. Do not include any white space in your answer.

SUBC(R31, Rx, Ry) ✖ Answer: SUB(R31,Rx,Ry)

If best implementation is CTL, select the appropriate value for each control signal, otherwise select NONE for each control signal.

Instr	ALUFN	WERF	BSEL	WDSEL	MOE	MWR	RA2SEL	PCSEL	ASEL	WASEL
NEG	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE
	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE	✔ Answer: NONE

Explanation
The NEG operation can be implemented as a macro that subtracts Rx from R31 and stores the results in Ry.

3. A PC-relative Store instruction:

```
STR(Rx, C )
  EA ← PC+4+4*SEXT(C)
  Mem[EA] ← Reg[Rx]
  PC ← PC + 4
```

Best implementation strategy

- ○ Macro
- ◉ CTL
- ○ Hardware

✔

If best implementation is Macro, enter the equivalent instruction, otherwise enter NONE. Do not include any white space in your answer.

NONE ✔ Answer: NONE

If best implementation is CTL, select the appropriate value for each control signal, otherwise select NONE for each control signal.

Instr	ALUFN	WERF	BSEL	WDSEL	MOE	MWR	RA2SEL	PCSEL	ASEL	WASEL
STR	A	0	-	-	0	1	1	0	1	-
	✔ Answer: A	✔ Answer: 0	✔ Answer: -	✔ Answer: -	✔ Answer: 0	✔ Answer: 1	✔ Answer: 1	✔ Answer: 0	✔ Answer: 1	✔ Answer: -

Explanation
There is no existing instruction that performs a PC-relative Store in one cycle. However, the datapaths for computing the desired EA = PC+4+4*SEXT(C) are available. You just need to set ASEL = 1 and ALUFN = A. The remaining control signals are set up to perform a ST operation, so WERF = 0, RA2SEL = 1, MOE = 0, and MWR = 1. BSEL, WDSEL, and WASEL are all don't cares.

4. An instruction that computes $\overline{X} \cdot Y$, for X in Rx and Y in Ry.

```
BITCLR(Rx, Ry, Rz)        // clear selected bits

  Reg[Rz] ← ~Reg[Rx] & Reg[Ry]   // (AND Ry with complement of Rx)
  PC ← PC + 4
```

Best implementation strategy

- Macro
- CTL
- Hardware

✔

If best implementation is Macro, enter the equivalent instruction, otherwise enter NONE. Do not include any white space in your answer.

NONE ✔ Answer: NONE

If best implementation is CTL, select the appropriate value for each control signal, otherwise select NONE for each control signal.

Instr	ALUFN	WERF	BSEL	WDSEL	MOE	MWR	RA2SEL	PCSEL	ASEL	WASEL
BITCLR	100100	1	0	1	-	0	0	0	0	0
	✔ Answer: 100100	✔ Answer: 1	✔ Answer: 0	✔ Answer: 1	✔ Answer: -	✔ Answer: 0	✔ Answer: 0	✔ Answer: 0	✔ Answer: 0	✔ Answer: 0

Explanation

Set up the boolean operators so that they execute a $\overline{X} \cdot Y$ function. This is done by setting the two most significant bits of ALUFN to 10 to specify that its a boolean operation, and then setting the remaining four bits so that the desired boolean operation whose truth table is:

Y	X	
0	0	0
0	1	0
1	0	1
1	1	0

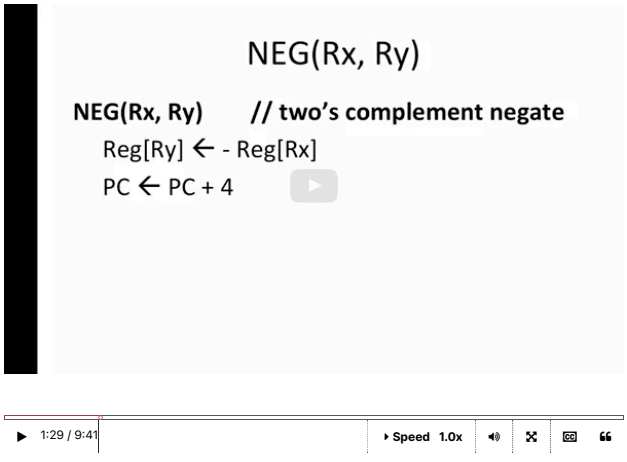
This can be implemented by modifying the CTL ROM to create a new boolean operation with this truth table.
Since the encoding of the bottom 4 bits of ALUFN is abcd where *a* corresponds to both inputs being 1 and *d* corresponds to both inputs being 0, then setting abcd to 0100 results in the desired function being implemented. So the 6 bit ALUFN is 100100. The remaining control signals follow the pattern of all other basic ALU operations (e.g., AND).

Submit

Answers are displayed within the problem

A Better Beta

Start of transcript. Skip to the end.



1:29 / 9:41 Speed 1.0x

In this problem, we are going to consider several instructions that we want to add to our beta.

For each of these instructions, we will need to decide what the minimum requirement is to add that instruction.

The simplest addition would be a macro that references a single already existing beta instruction.

If our new instruction cannot be implemented by simply defining a macro, then we want to consider whether adding a new opcode, and producing the appropriate control ROM signals for it, will enable the new operation to be executed on our existing Beta datapaths.

Video
Download video file

Transcripts
Download SubRip (.srt) file
Download Text (.txt) file

Discussion

Topic: 13. Building the Beta / WE13.1

Hide Discussion

Add a Post

Show all posts by recent activity

[Staff] Is part D correct? 9
We are told that the ALUFN for the instruction that computes not(X) · Y in the final part of these worked examples is 100100. The explanation is familiar territory fr...

Value of MOE signal in part C. STR instruction. 8

I did not understand in the video explanation why MOE is 0 and not dont care. Could somebody explain?