LSU EE 4720 Homework 4 Solution Due: 15 March 2021

☣ Do not hand in paper copies. Instead, E-mail your solution to koppel@ece.lsu.edu. The preferred format is a PDF file.

Problem 1: Recall that code for the solution to Homework 2 included a loop that traversed a tree. The decision on whether to descend to the left or right child of a node was based on the next bit of compressed text. Several instructions were devoted to testing that next bit, and to checking whether a new word of bits needed to be loaded. In this assignment we are going to add a new instruction, `bnbb` (branch next bit big-endian), to MIPS that will allow such code to be written with fewer instructions.

Instruction `bnbb rV, rP, TARG0, TARG1` works as follows. Register `rV` holds a bit vector, and register `rP` holds a position in the bit vector. (A bit vector is just a number, but it's called a bit vector when we are interested in examining specific bits in the number's binary representation.) If the value of `rP` is `0` then it refers to the MSB of `rV`, if the value of `rP` is `1` it refers to position 1 (to the right of the MSB), etc. Let `pos` refer to bits `5:0` of `rP`. If `pos` is in the range 0 to 31 (inclusive) then `bnbb` will be taken, otherwise (values from 32 to 63) `bnbb` is not taken. When `bnbb` is taken it will branch to `TARG0` if bit `pos` in `rV` is `0` and to `TARG1` if bit `pos` in `rV` is `1`. Regardless of whether `bnbb` is taken register `rP` is written with `rP+1`. See the code and comments below:

```
# With sample values below bnbb is taken to LCHILD since bit 30 of 0x5 is zero.
# $t8 = 0x5 (bit vector),  $t9 = 30 (pos)
bnbb $t8, $t9, LCHILD, RCHILD
addi $v0, $t9, 0               # Delay slot insn. Here t9 is 31.

# This code is only executed when $t9 in range 32-63 before bnbb executes.
# Fall through. Updates t8 and t9
addi $t6, $t6, 4        # Update address ..
lw $t8, 0($t6)          # No more bits, load a new word.
addi $t9, $0, 0
```

The `bnbb` instruction can be used to eliminate at least two instructions in the `hw02` solution. First, there would no longer be a need to shift the bit vector (the `sll $t8, $t8, 1` instruction). Instead, the `bnbb` instruction would automatically increment a bit position register. Also, there would no longer be a need for a second branch to check whether all 32 bits in the bit vector were examined. (That was the `bne $a1, $t9, EXAMINE_NEXT_BIT` instruction.)

In the subproblems below complete the specification for `bnbb` and show hardware to implement it.

An Inkscape SVG version of the hardware diagram can be found at
`https://www.ece.lsu.edu/ee4720/2021/hw04-br-3way.svg`.

(*a*) The description above leaves out a few details. In this problem fill them in. It may be helpful to attempt a solution to the next parts before answering this part.

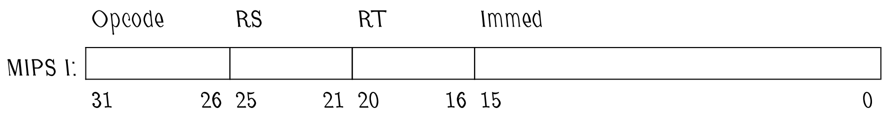
Show a possible encoding for `bnbb`. That possible encoding must be based on format I. Show how the two targets are specified and and whether `rV` is encoded in the `rt` or `rs` fields.

The format appears below. The `RT` register field is used for `rP` and the `RS` for `rV`. Placing `rP` in the `RT` field simplifies the decoding logic by a small amount. That's because the hardware needs to write back the incremented value of `rP` and there is already logic that sets the `dst` control signal to the `RT` field in format I instructions.

As with other branches, the `bnbb` will use displacement addressing, but unlike other branches there are two displacements (one displacement for `TARG0` and one displacement for `TARG1`). To accommodate two displacements the immediate field has been split in two as shown below. Note that with just eight bits of displacement the targets must be within 128 instructions of the branch (128 before the delay slot or 127 after the delay slot).

	Opcode	RS	RT	Imm0	Imm1
MIPS Ib:	0x2d	rV	rP		
	31 26	25 21	20 16	15 8	7 0

Here is the original MIPS format I:



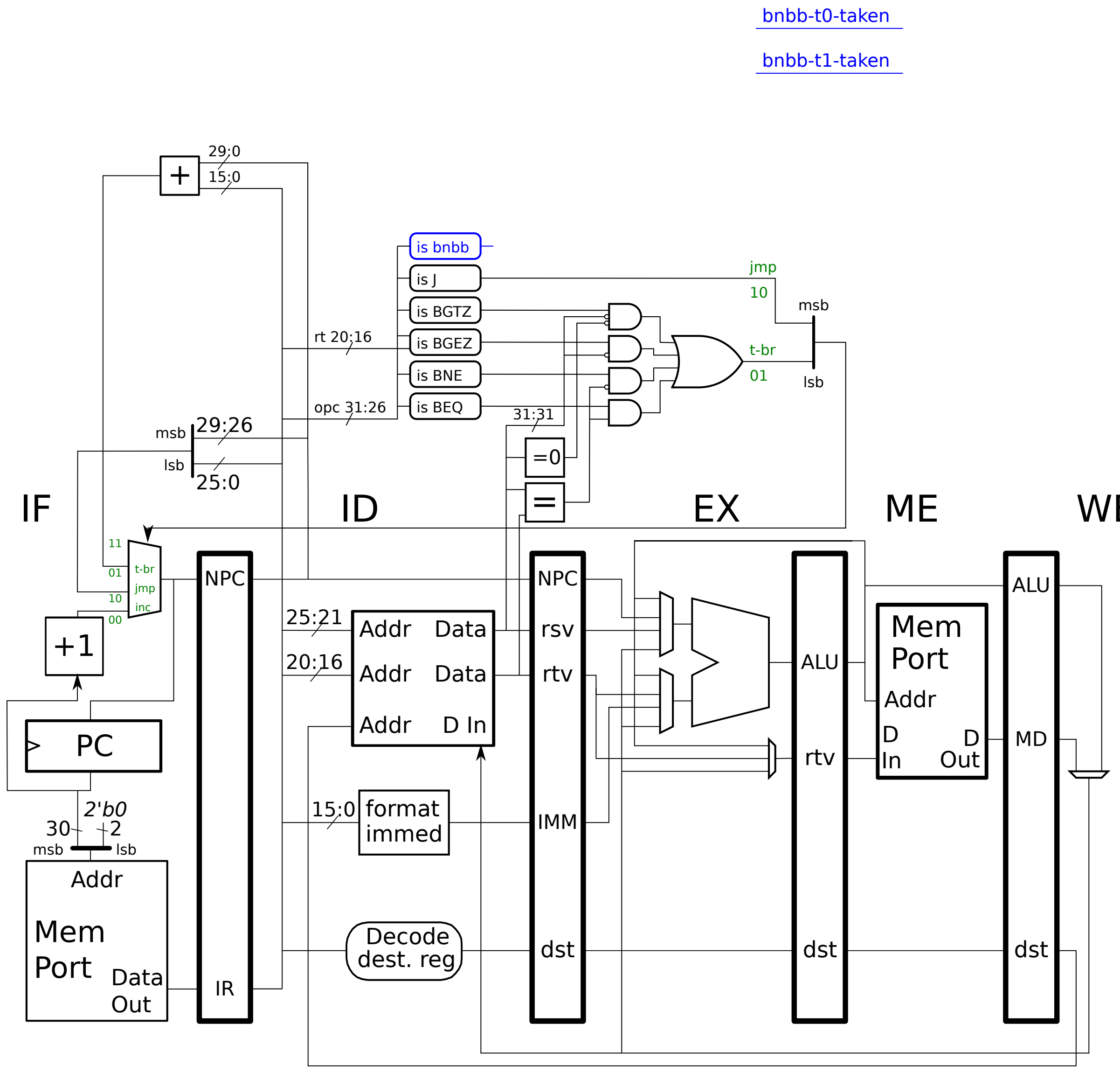
(*b*) For `bbnb` to work correctly the `rP` register value needs to be incremented. It would be nice if an existing ALU operation could do that. Explain why the `add` operation, used for the `add`, `addi`, `lw`, and other instructions, would not work.

The `bbnb` instruction needs to add a 1 to the `rP` value. The ALU `add` operation used for those instructions computes the sum of the upper and lower ALU inputs. One of those inputs would be the `rP` value, but there is no way to set the other ALU input to 1 with the illustrated hardware. The `add` operation could be used if a new input were added to one of the ALU multiplexors and that input was set to the constant 1.

(*c*) The diagram below shows a five-stage MIPS implementation including some branch hardware. Also shown is logic to detect the `bnbb` instruction and two placeholder wires, `bnbb-t0-taken` and `bnbb-t1-taken`. Wire `bnbb-t0-taken` should be set to `1` if there is a `bnbb` in the `ID` stage and it should be taken to `TARG0`. The definition of `bnbb-t1-taken` is similar. If there is not a `bnbb` in `ID` or if there is and it's not taken, then both wires should be `0`.

In this problem design the logic to drive those wires. (The solution to this and the following problem can be done on the same diagram, or on separate diagrams.)

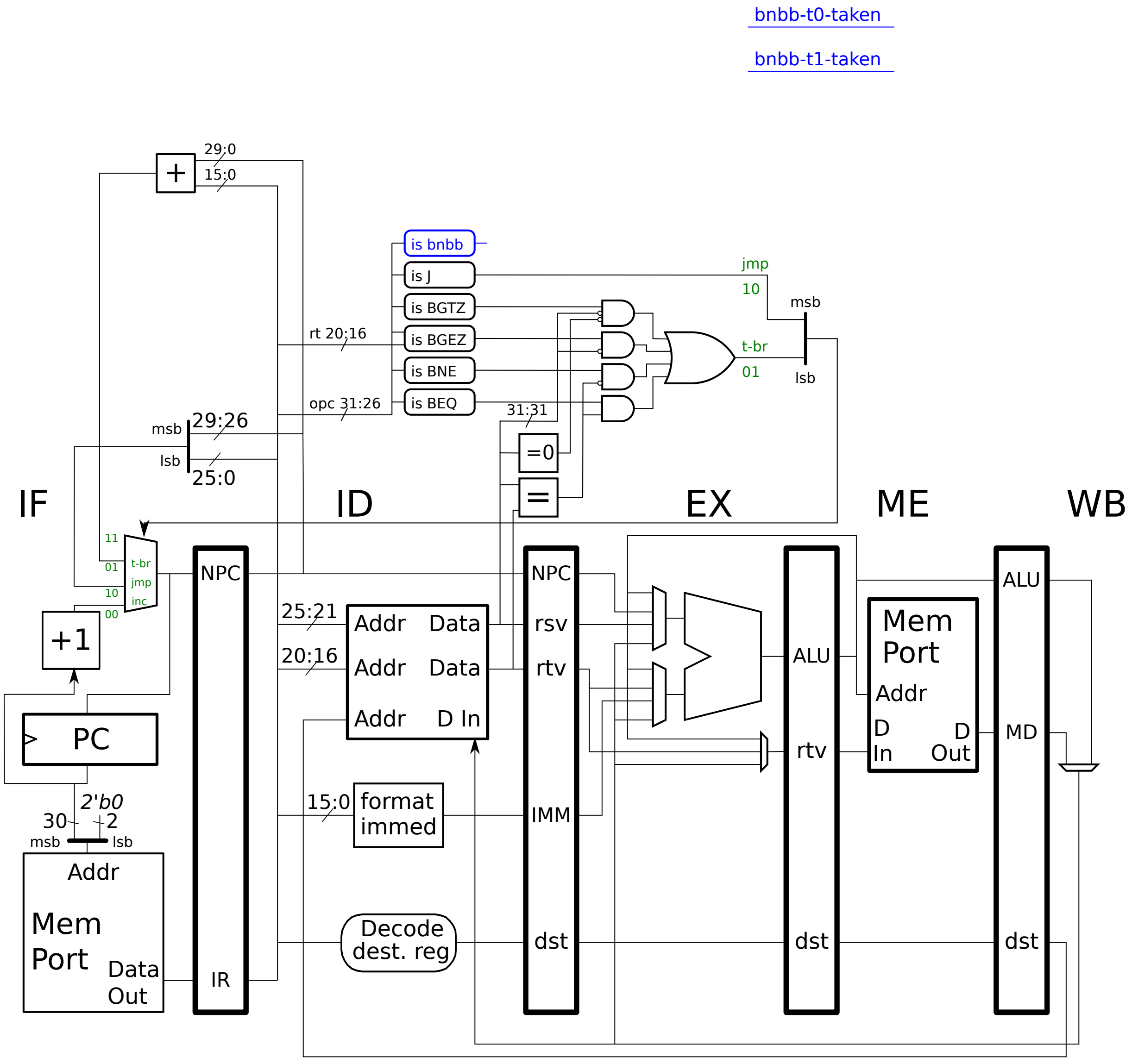
The solution appears after part d.



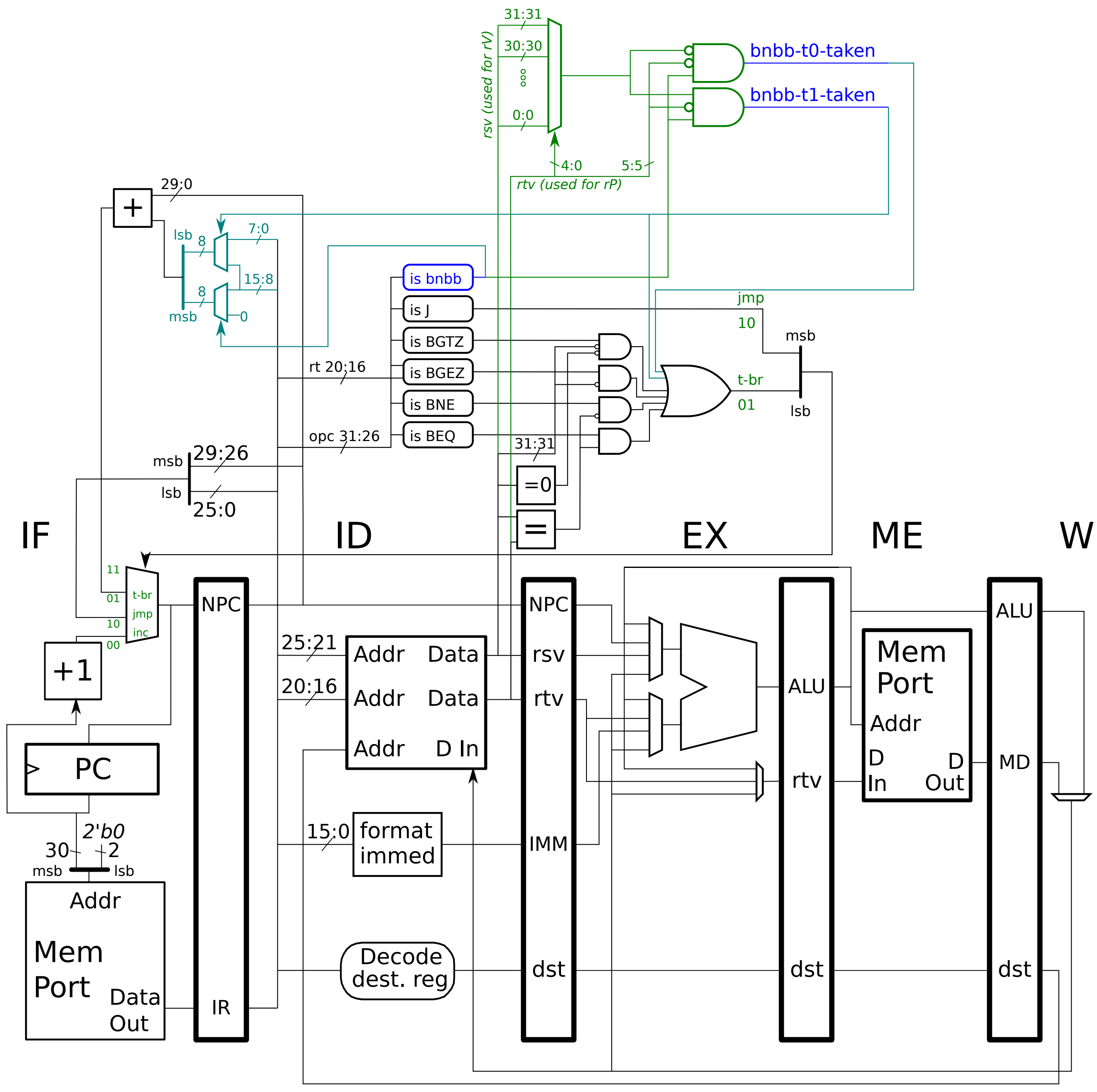
(*d*) Modify the hardware below so that when `bnbb-t0-taken` is `1` target `TARG0` is used and when `bnbb-t1-taken` is `1` target `TARG1` is used. Follow the points below.

- Design for lower cost rather than higher performance.
- There is an unused input on the `PC` mux. That can be used, but does not have to be used.
- As always, hardware must be reasonably efficient.
- As always, do not break other instructions.

The solution is on the next page.



The solution for parts c and d appears below. A multiplexor extracts the needed bit from **rsv**. To keep costs low **bnbb** computes the target address using the same adder as other branch instructions.


31:31
30:30
0:0
rsv (used for rV)
bnbb-t0-taken
bnbb-t1-taken
4:0
5:5
rtv (used for rP)
29:0
+
lsb
8
7:0
15:8
msb
0
is bnbb
is J
is BGTZ
is BGEZ
is BNE
is BEQ
jmp
10
msb
rt 20:16
t-br
01
lsb
opc 31:26
31:31
=0
=
msb
29:26
lsb
25:0
IF
ID
EX
ME
WB
11
01
10
00
t-br
jmp
inc
NPC
+1
25:21
Addr
Data
20:16
Addr
Data
Addr
D In
NPC
rsv
rtv
ALU
ALU
Mem
Port
Addr
D
In
D
Out
rtv
MD
PC
2'b0
30
2
msb
lsb
Addr
Mem
Port
Data
Out
IR
15:0
format
immed
IMM
Decode
dest. reg
dst
dst
dst