ECPE 173 – Spring 2013
Lab Project 5

In this lab project, we will extend our Beta processor to support branches, jumps, exceptions, and interrupts. Our basic design looks like Figure 1.

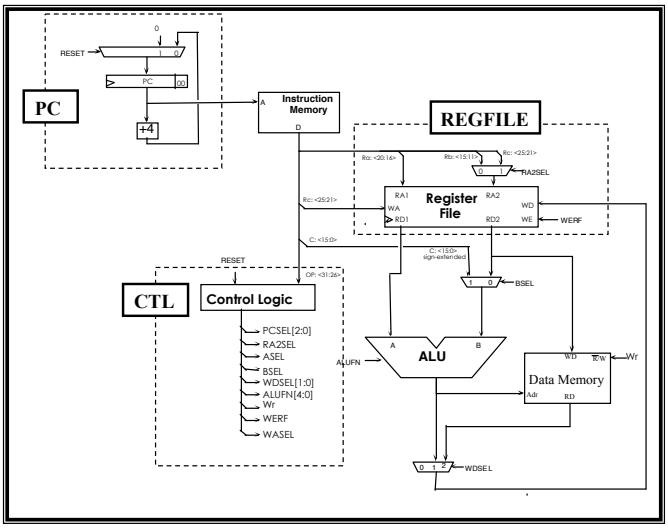

Figure 1: Basic Beta Processor

Our expanded design will look like Figure 2.

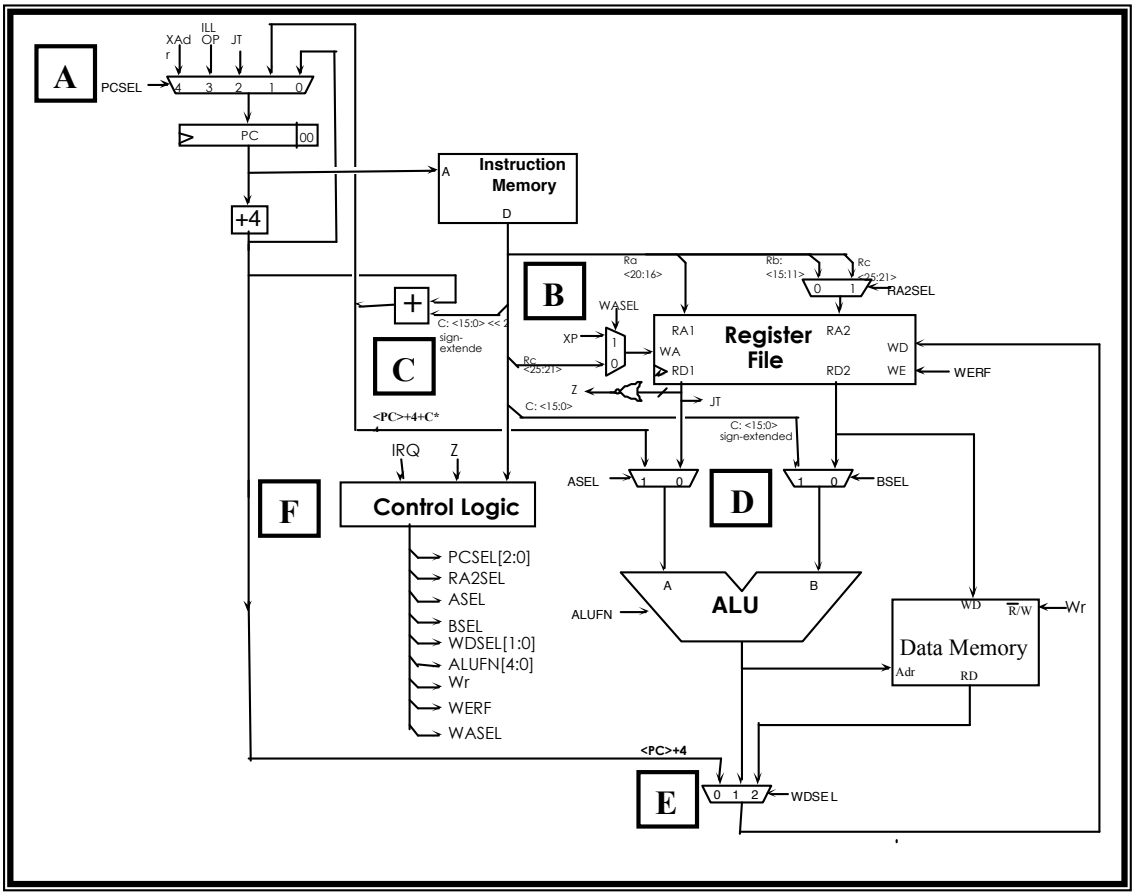

Figure 2: Expanded Beta Design

1. Everyone needs to extend his or her own Lab Project 4. If you collaborate with anyone, you need to follow the collaboration policy. No collaboration on the writing is allowed – any duplicate sentences/paragraphs will result in a zero.

Read this complete lab project. We are adding some tricky logic so make sure you understand the ideas behind that logic before beginning. All Beta documentation is in the Lab Project 4 folder on Sakai.

Start a new Word document report for this project. Create appropriate document headings and write a short Problem section describing the goals of this lab (in your own words – not mine!).

3. Create a new folder for this lab and copy your lab project 4 files there. Do not overwrite the original lab project 4. Download checkoff files from Sakai->Resources->Lab Projects->Project 5.

4. Review the bulleted list of design features and notes regarding what you will add to your Beta design. Determine the order which you will implement the new features. In your Word document, create a "Design Approach" section and describe how you will approach the design.

5. Design the system to include the new features. You need to modify the subcircuit definition slightly to:

```
.subckt beta clk reset irq ia[31:0] id[31:0] ma[31:0] moe mrd[31:0]
+ wr mwd[31:0]
        ... implementation here ...
.ends
```

To verify your implementation, use:

```
.include "lab5checkoff.jsim"
```

6. In your Word document, create an "Implementation" section and describe how you implemented your design, any debugging you had to perform, and modifications to your original design.

NOTES AND FEATURES

The first set of notes relate to the letters in Figure 2.

A. While not indicated, you do still need to reset the PC to zero. `XAdr` and `ILL_OP` represent constant addresses used when Beta sees an interrupt or executes and instruction with an illegal or unimplemented opcode. For this project, assume `XAdr=8` and `ILL_OP=4`. The checkoff file will set the first three locations of instruction memory to contain branch instructions to jump to code dealing with reset, illegal instructions, and interrupts.

See Appendix A for information regarding the supervisor bit.

B. The WASEL multiplexer determines the write address for the register file.

C. The branch offset adder adds PC+4 to the immediate field. Remember that the 16-bit immediate field needs to be word-aligned (so multiply or shift or wire) and then sign-extended.

D. ASEL multiplexer and Z logic connect to the output of the first source port. This port also connects directly to the JT inputs of the PCSEL multiplexer (remember to word-align and connect the supervisor bit as bit 31).

E. To include branches and jumps, we need to save off PC+4 to a register (usually $ra, but in this case XP), which requires expanding the WDSEL multiplexer. The supervisor bit is also saved as part of this.

F. Control logic needs to expand to support your design. This expansion requires new control signals:

PCSEL[2:0] – this will require using the Z signal for branches
ASEL
WDSEL[1:0]
WASEL

GENERAL NOTES

- This version must now support LDR, JMP, BNE, and BEQ instructions. The multiply and divide instructions are optional; otherwise the complete Beta instruction set should be implemented. Unimplemented instructions should cause an exception by choosing the correct PCSEL multiplexer input.
- An interrupt-request (IRQ) input has been added to the Beta. When this signal is 1 and the Beta is in "user mode" (PC31=0), an interrupt should occur. Interrupts should not occur when in "supervisor mode" (PC31=1). For interrupts occurring in "user mode," you need to add logic that causes the Beta to abort the current instruction, save the current PC+4 in register XP, and set the PC to 0x80000008. Additionally, you should make sure to set WR=0 to avoid writing anything to memory if the instruction aborted is a store instruction.
- On reset, the PC=0x8000000 and WR=0.

APPENDIX A – SUPERVISOR BIT

Note on supervisor bit: The high-order bit of the PC is dedicated as the "Supervisor" bit (see section 6.3 of the Beta Documentation). Instruction fetch and the LDR instruction ignore this bit, treating it as if it were zero. The JMP instruction is allowed to clear the Supervisor bit or leave it unchanged, but cannot set it, and *no other instructions may have any effect on it*. Only reset, exceptions and interrupts cause the Supervisor bit to become set. This has the following implications for your Beta design:

1. 0x80000000, 0x80000004 and 0x80000008 are loaded into the PC during reset, exceptions and interrupts respectively. This is the only way that the supervisor bit gets set. Note that after reset the Beta starts execution in supervisor mode.

2. Bit 31 of the PC+4 and branch-offset inputs to the PCSEL mux should be connected to PC31, i.e., the value of the supervisor bit doesn't change when executing most instructions.

3. You'll have to add logic to bit 31 of the JT input to the PCSEL mux to ensure that JMP instruction can only clear or leave the supervisor bit unchanged. Here's a table showing the new value of the supervisor bit after a JMP as function of JT31 and the current value of the supervisor bit (PC31):

old PC31	JT31	new PC31
0	--	0
1	0	0
1	1	1

4. Bit 31 of the branch-offset input to the ASEL mux should be set to 0 – the supervisor bit is ignored when doing address arithmetic for the LDR instruction.

5. Bit 31 of the PC+4 input to the WDSEL mux should connect to PC31, saving the current value of the supervisor whenever the value of the PC is saved by a branch instruction or trap.