6.004 Computation Structures
Spring 2009

Pipelined Beta

Problem 1. Beta quickies.

A. ★ In a 5-stage pipelined Beta, when does the hardware use its ability to insert NOP into the instruction stream at the IF stage (using the MUX controlled by Annul^{IF})?

A NOP is inserted in the IF stage when annulling instructions in the branch delay slot because of a taken branch or when flushing the pipeline due to a fault in the ALU or MEM stages.

B. ★ In a 5-stage pipelined Beta, when does the hardware use its ability to insert a NOP into the instruction stream at the ALU stage (using the MUX controlled by Annul^{ALU})?

If a fault occurs in the MEM stage (eg, illegal memory address), the instruction currently in the ALU stage needs to be discarded, which the hardware does by substituting a NOP. Note that a fault in the MEM stage discards all the instructions currently in the IF, RF and ALU stages since they logically come *after* the instruction which caused the fault.

C. ★ Ben Bitdiddle is thinking about modifying a 5-stage pipelined Beta to add a "Jump if Memory Zero" instruction (JMZ) that fetches the contents of a memory location and jumps if the fetched value is zero. How many branch delay slots would follow a JMZ instruction in the modified 5-stage pipelined Beta?

The contents of the memory location accessed by JMZ aren't available until the instruction reaches the WB stage. If the branch decision is made at that point, instructions in MEM, ALU, RF and IF have already been fetched -- so there are 4 branch delay slots.

D. Suppose the following code were running on a Beta implementation with a 5-stage pipeline, full bypassing and 1 branch delay slot with annulment.

```
PUSH (R1)
PUSH (R2)
LD (BP, -12, R0)
LD (BP, -16, R1)
CMPEQ (R0, R1, R2)
BT (R2, L1)
```

When the CMPEQ is executed, assuming no interrupts, where does the value for R0 come from? How about the value for R1? (The choices would be from the register file or bypassed from one of the pipeline stages.)

The CMPEQ instruction stalls in the RF stage until values for both R0 and R1 are available. The value for R1 isn't available until the second LD reaches the WB stage at which point the results from the first LD have already been written into the register file. So R0 comes from the register file and R1 is bypassed from the WB stage.

E. ★ Which of the following pipeline hazards cannot be dealt with transparently and at no performance cost by bypassing?
   A. A shared register between consecutive ALU instructions.
   B. A BR followed by an ALU instruction using the BR.
   C. An LD followed by an ALU instruction using the LD.
   D. Access to LP by the first instruction in a called procedure.
   E. Access to XP by the first instruction in an interrupt handler.

(C) Since the data from an LD instruction isn't available until the WB stage, bypassing can't help.

F. ★ The number of branch delay slots reflects
   A. The distance between the instruction fetch stage and the stage at which the branch decision is made.
   B. The distance between the writeback stage and the stage at which the branch decision is made.
   C. The total length of the pipeline.
   D. The position within the pipeline of the instruction fetch stage.
   E. The number of cycles required for a fetch from data memory.

(A) The number of branch delay slots refers to the number of instructions following a branch that are fetched by the pipeline before the branch decision is made.

Problem 2. A common method for communicating with input and output devices is to assign them to one or more memory addresses. This technique is called memory-mapped I/O. Some I/O locations are used to address status words that indicate the availability of an associated I/O device. These status words indicate if an input device has new input information available, or if an output device has processed its previous output request. Often, computers will execute tight loops waiting for the status of an I/O device. Consider the following instruction sequence for checking the status of an external I/O device.

```
loop: LD(R31, status, R0)
      BEQ(R0, loop, R31)
      ADD (R0, R1, R2)
```

The following pipeline diagram illustrates the execution of this instruction sequence on a standard 5-stage pipelined Beta:.

IF	LD	BEQ	ADD	ADD	ADD	LD	BEQ	ADD	ADD	ADD
REG		LD	BEQ	BEQ	BEQ	NOP_3	LD	BEQ	BEQ	BEQ
ALU			LD	NOP_1	NOP_2	BEQ	NOP_3	LD	NOP_4	NOP_5
MEM				LD	NOP_1	NOP_2	BEQ	NOP_3	LD	NOP_4
WB					LD	NOP_1	NOP_2	BEQ	NOP_3	LD

A. ★ How many clock cycles does it take to execute one iteration of the 2-instruction loop given?

5 clocks. Measuring from the LD instruction of one iteration to the LD instruction of the next, we get five cycles.

B. ★ What aspect of the instruction sequence causes NOP1 to be inserted into the pipeline?

The RA field of the BEQ matched the RC field of the LD in the ALU stage. BEQ is testing the value in R0 which was the destination for the preceeding LD instruction. The memory data being accessed by LD won't be available to the Beta until the LD instruction reaches the WB stage. So the hardware let's the LD instruction progress through the pipeline while stalling the BEQ instruction in the RF stage until the necessary data is available. The "gap" that opens between the two instructions is filled with NOPs (in this case, NOP1 and NOP2).

C. ★ What aspect of the instruction sequence causes NOP2 to be inserted into the pipeline?

The RA field of the BEQ matched the RC field of the LD in the MEM stage.

D. ★ What aspect of the instruction sequence causes NOP3 to be inserted into the pipeline?

All taken BEQ instructions cause NOPs to be inserted. We have chosen to annul instructions that follow taken branches. This is accomplished by in the IF stage by replacing annulling instructions with NOPs.

E. In a non-standard version of the 5-stage pipelined Beta, **where the instruction following a branch is not annulled**, which of the following statements would be true?
   A. The ADD instruction would be executed each time through the loop.
   B. The loop would still take 5 cycles to execute
   C. The value of the register R0 that is tested by the BEQ instruction comes from a by-pass path
   D. The value of the register R0 that is accessed by the ADD instruction comes from the register file.

All of the above.

Problem 3. The 5-stage pipelined Beta (as shown in lecture) is executing the sequence

```
ADD(R31,R31,R31)     | NOP
ADD(R1,R2,R1)
LD(R1,4,R1)
SUB(R1,R5,R6)
ORC(R1,123,R1)
SHL(R1,R1,R1)
```

A. ★ Which input is selected by the Ra bypass MUX when the ADD instruction is in the ALU stage?

The ALU output. The LD instruction is in the RF stage and needs to get the result of the ADD.

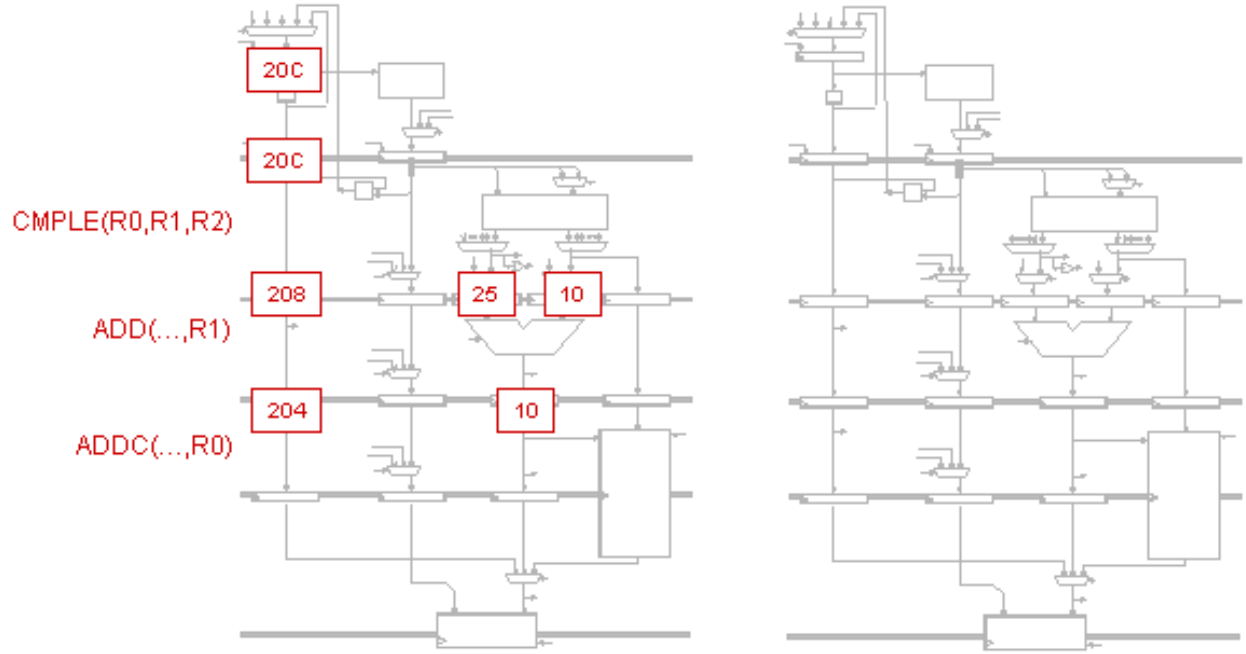
B. ★ Which input is selected by the Ra bypass MUX when the LD instruction is in the WB stage?

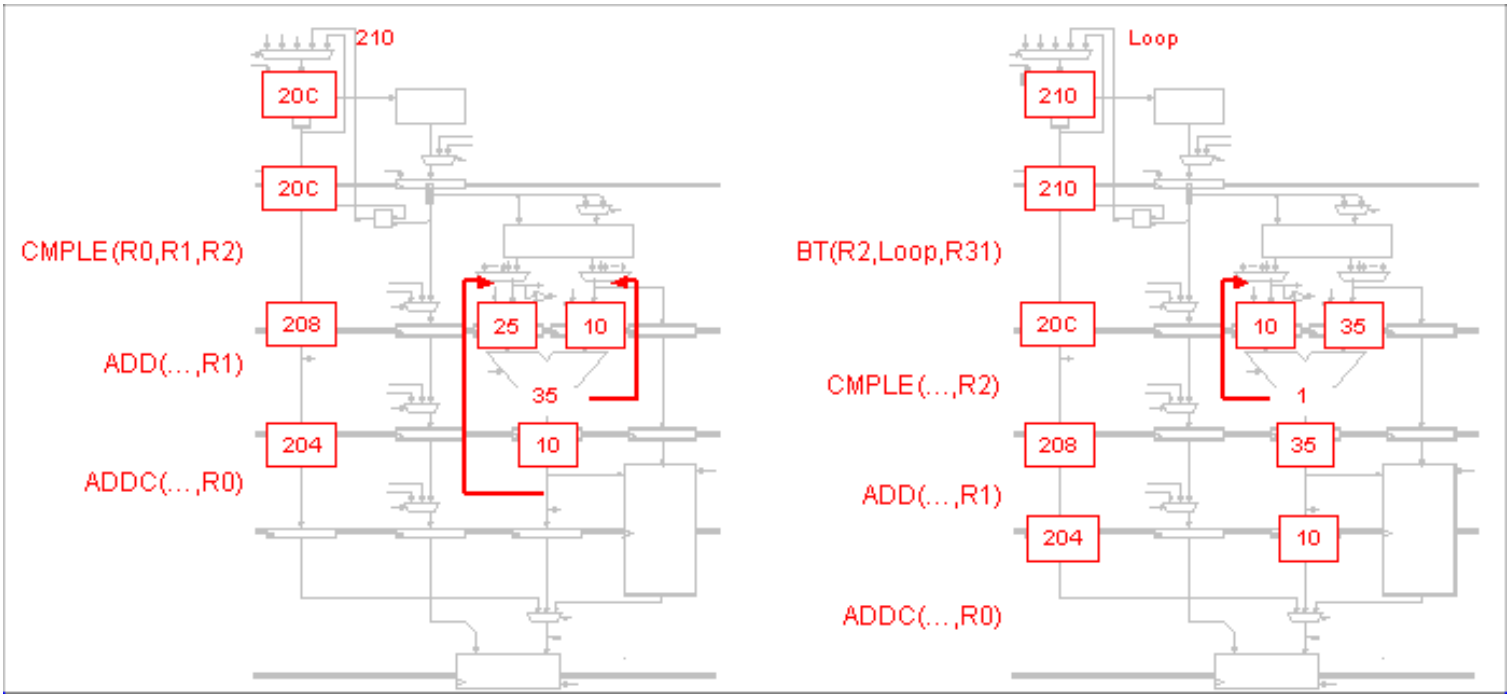
The WD input to the register file. After several stalls (holding SUB in the RF stage) LD's output is bypassed back to the RF stage.

Problem 4. ★ Each of the following scenarios shows a snapshot of a 5-stage Beta executing a sample code sequence. For each scenario, indicate the appropriate settings for the bypass muxes, the IR muxes, and the IR/ALU regs load enable signals. Then draw another snapshot showing the state of the 5-stage Beta on the following cycle.

A. **Scenario 1:** assume R2 contains 25

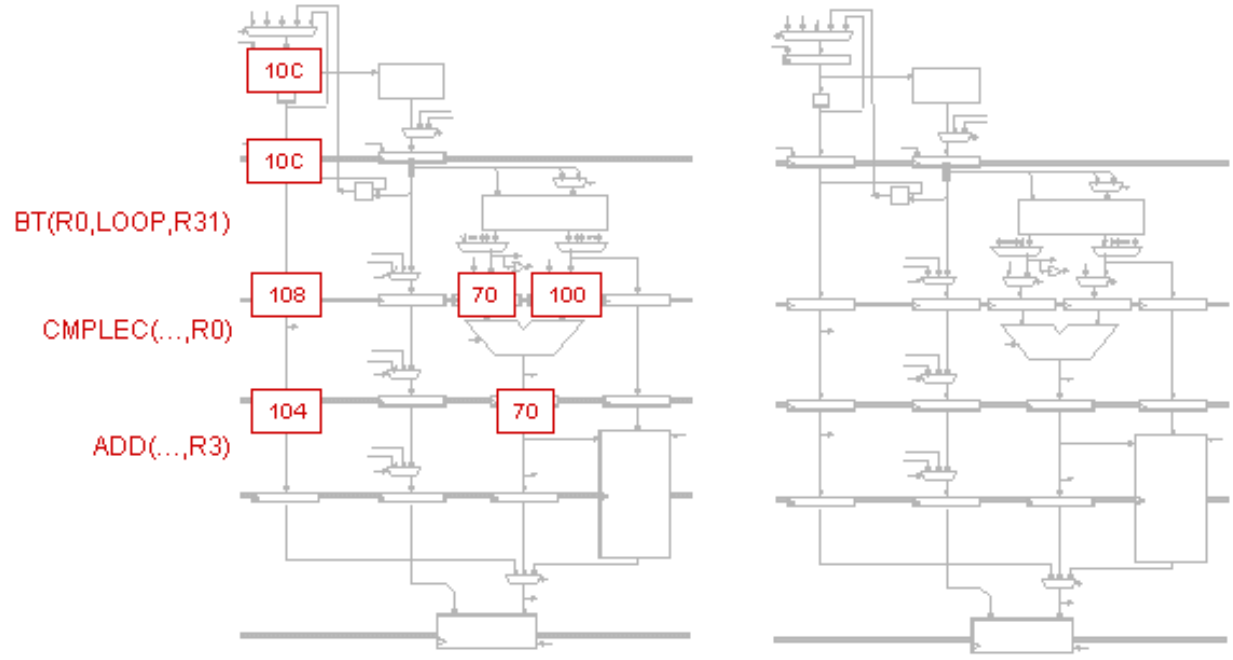
```
. = 0x200
ADDC(R31,10,R0)
ADD(R2,R0,R1)
CMPLE(R0,R1,R2)
BT(R2,Loop,R31)
```

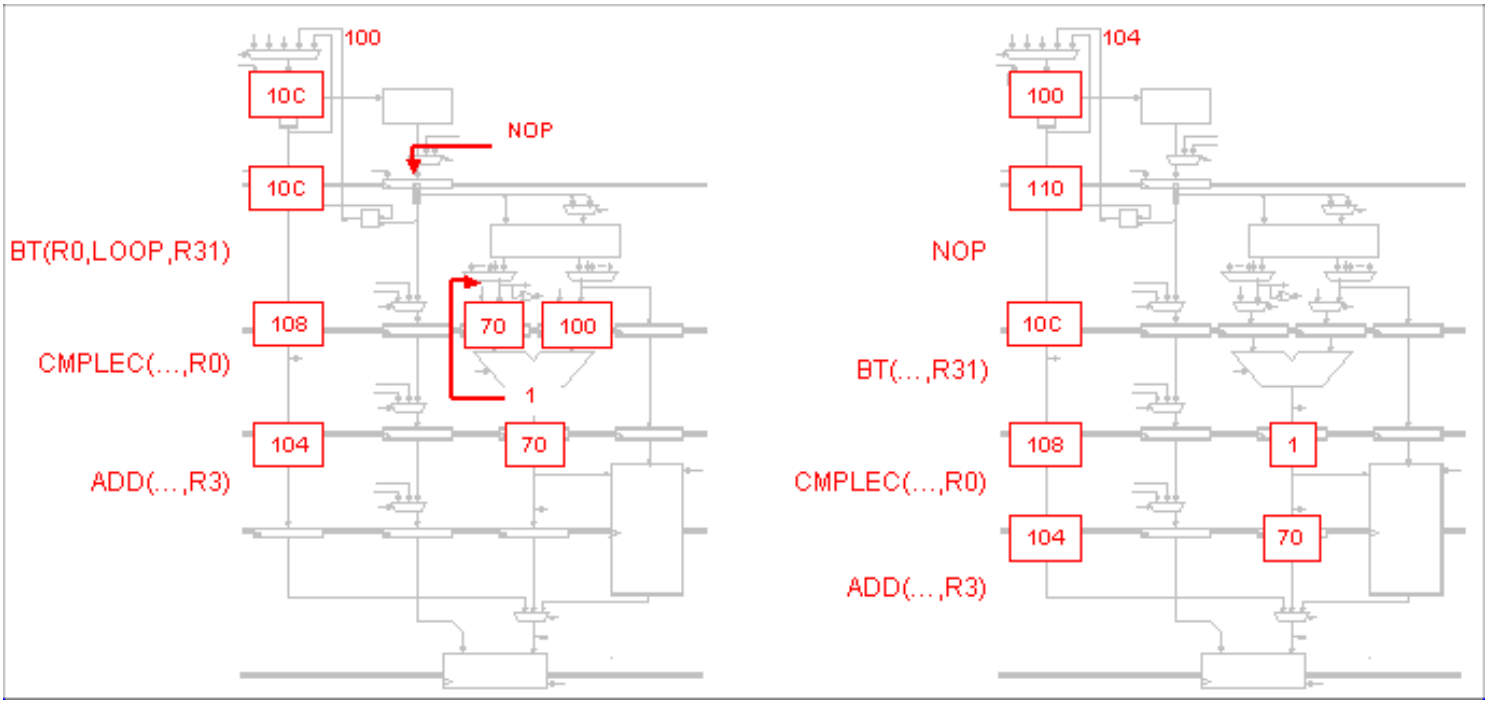




B. **Scenario 2:** assume R1 contains 10, R2 contains 60

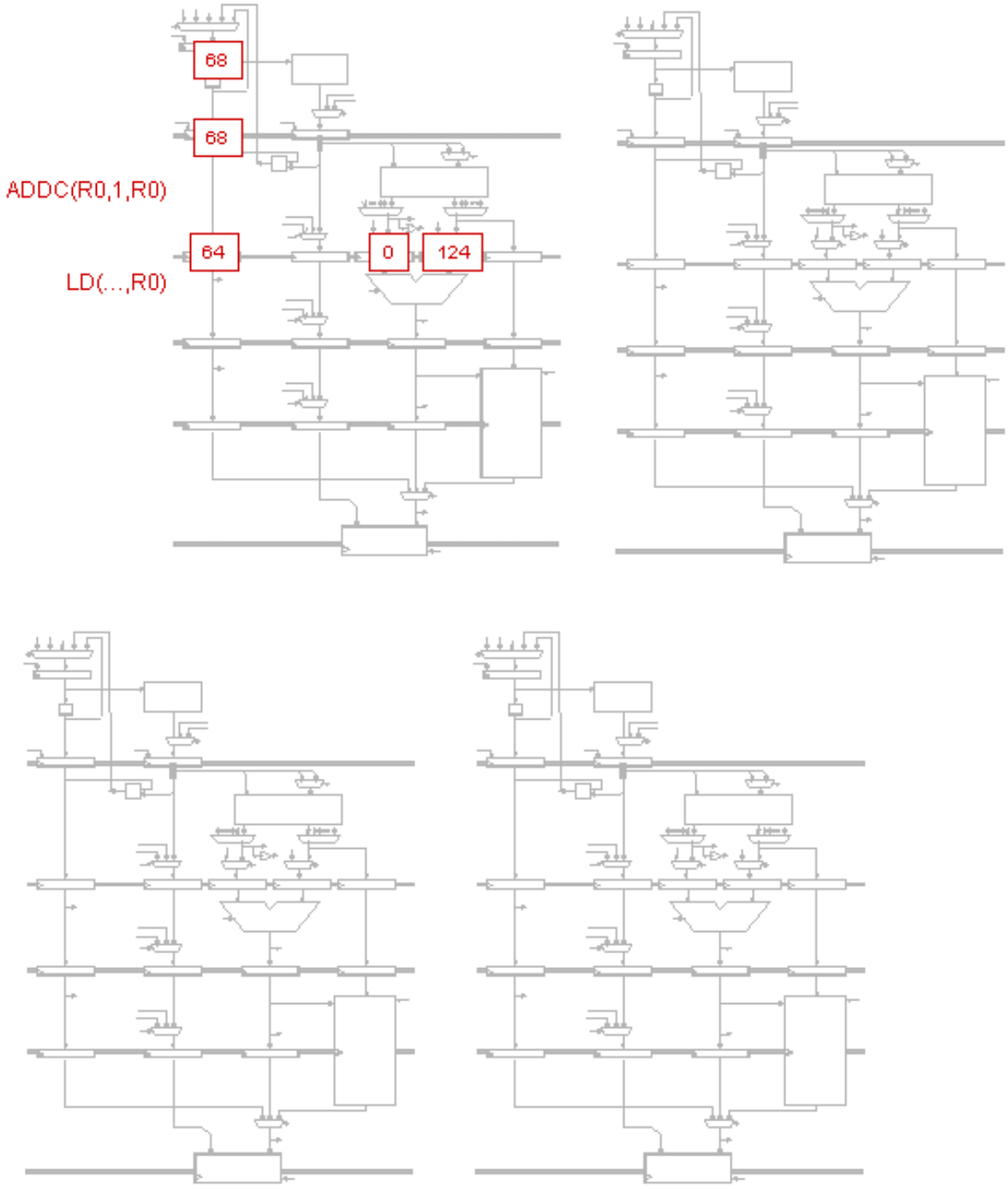
```
. = 0x100
LOOP: ADD(R1,R2,R3)
      CMPLEC(R3,100,R0)
      BT(R0,Loop,R31)
      SHLC(R3,1,R3)
```

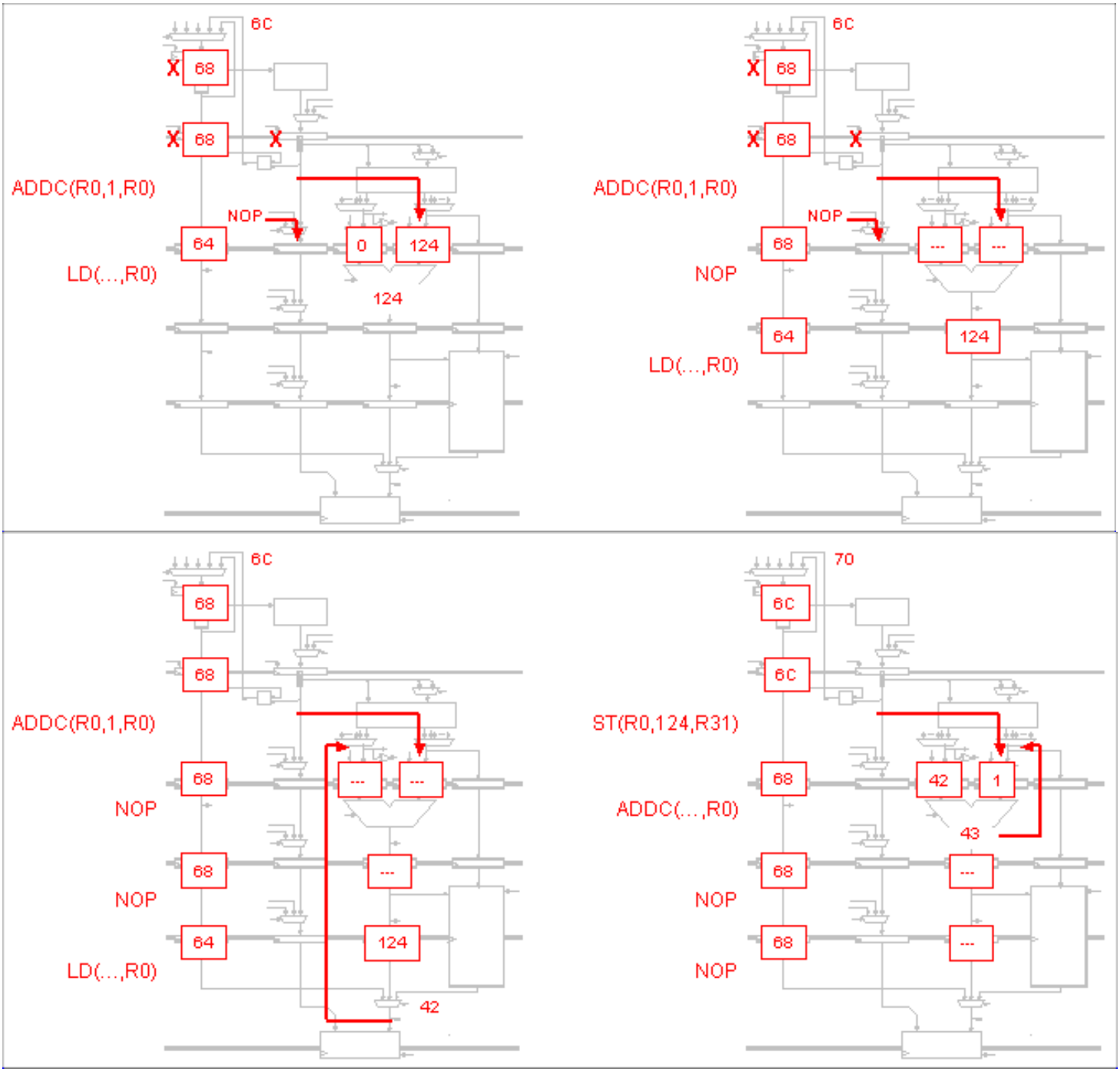




C. **Scenario 3:** (show 3 cycles of snapshot) assume Mem[124] contains 42

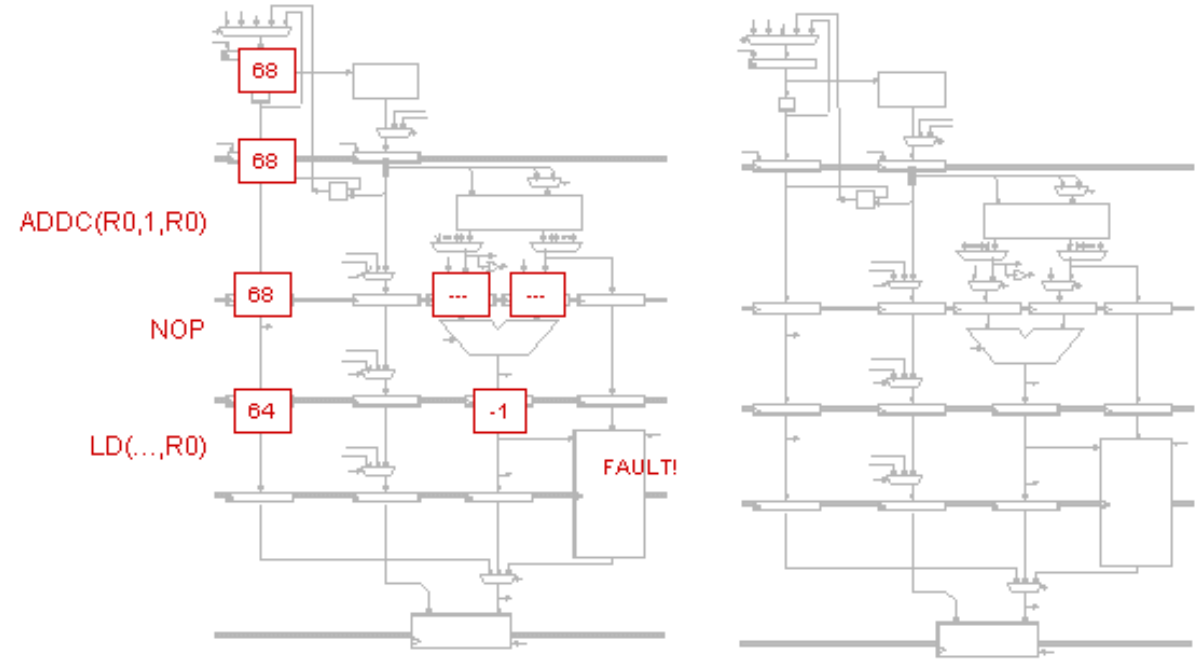
```
. = 0x60
LD(R31,124,R0)
ADDC(R0,1,R0)
ST(R0,124,R31)
```

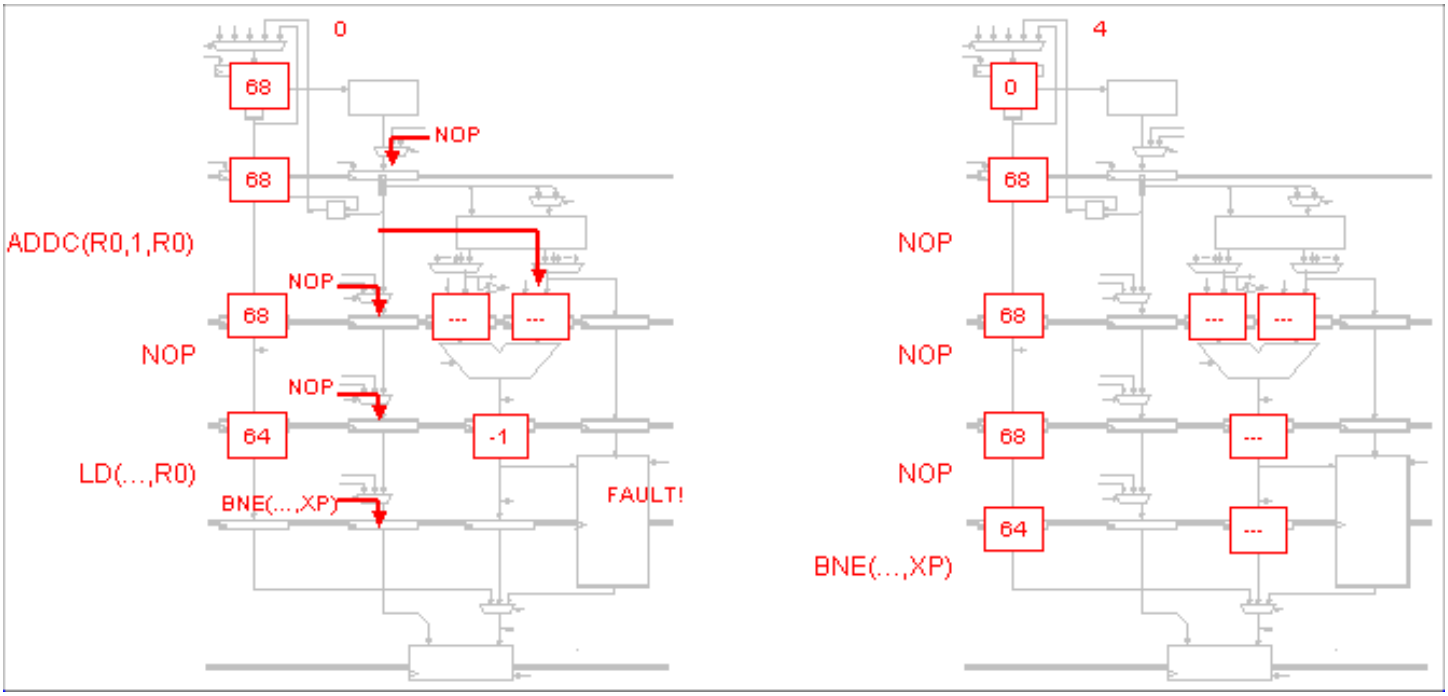

68
68
ADDC(R0,1,R0)
64
0
124
LD(…,R0)



D. **Scenario 4:** Show what happens when LD gets a MEMORY FAULT and is aborted in the MEM pipeline stage.

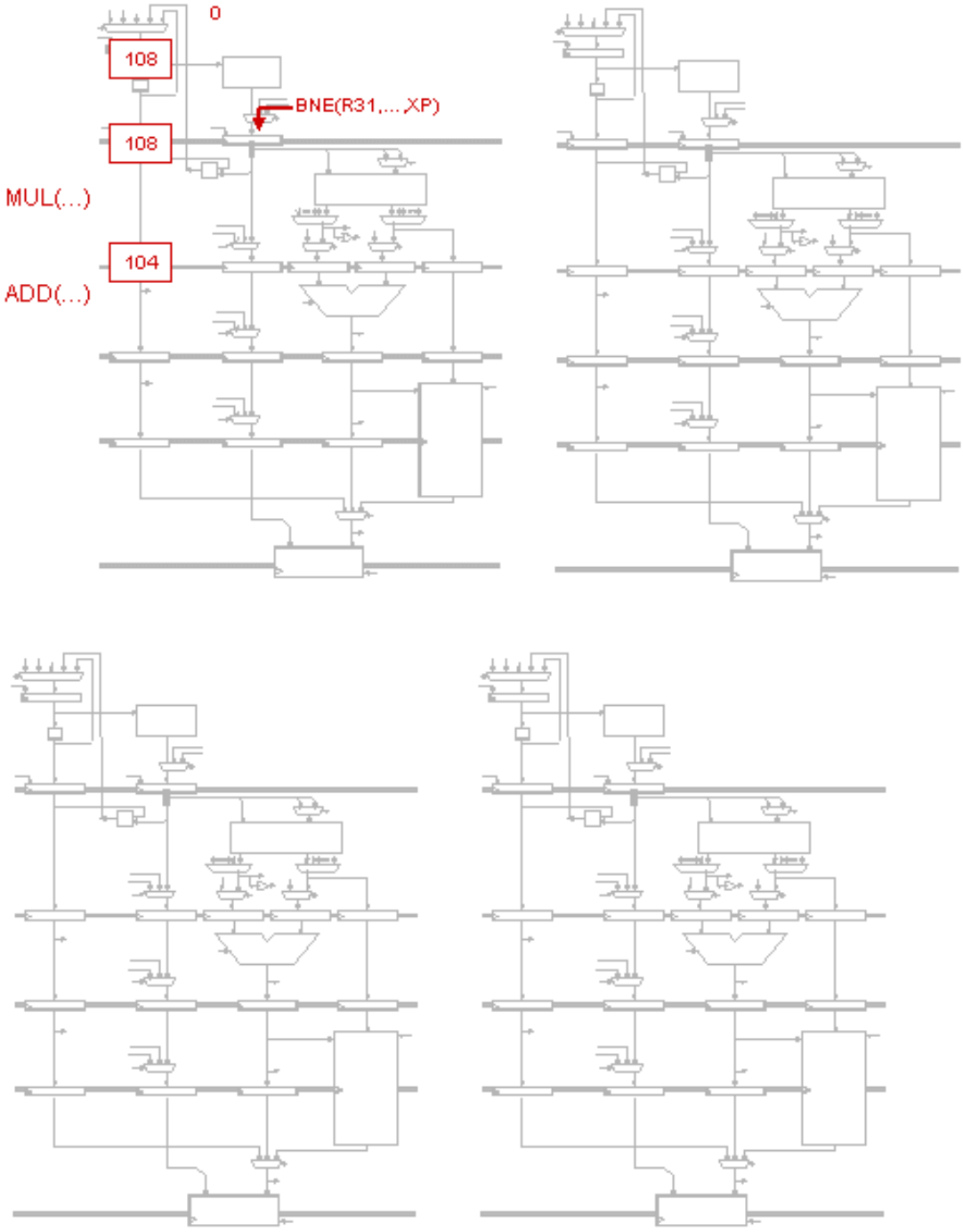
```
. = 0x60
LD(R31,-1,R0)
ADDC(R0,1,R0)
```

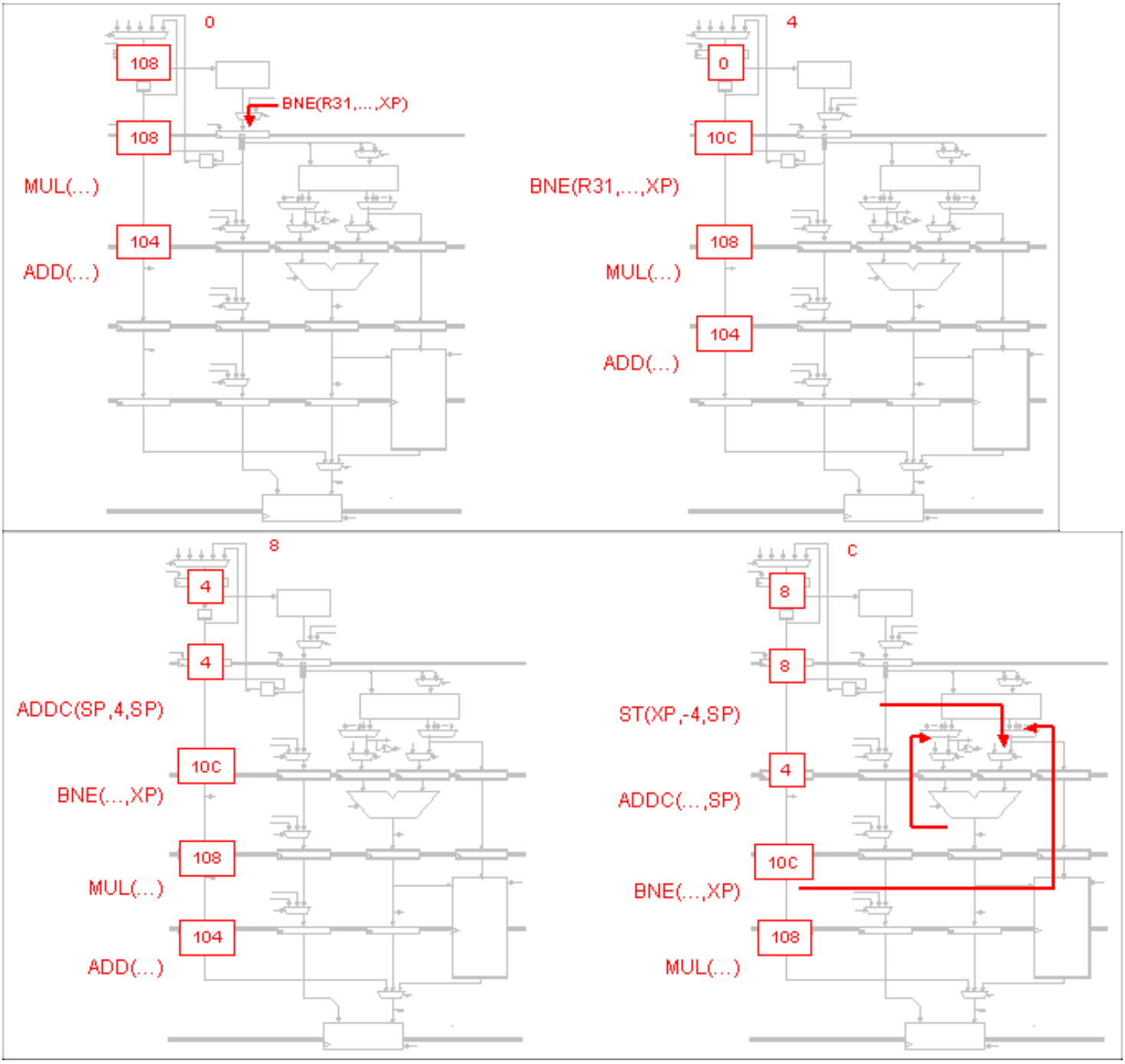




E. **Scenario 5:** (show 3 cycles of snapshot) Show what happens when an interrupt occurs when the Beta is fetching the SUB instruction. Assume the hardware sets PC^{IF} to 0 when taking an interrupt.

```
. = 0x100                          . = 0x0
ADD(...)                 IHANDLER: ADDC(SP,4,SP)   | PUSH(XP)
MUL(...)                           ST(XP,-4,SP)
SUB(...)  | Interrupt here         ...
```


0
108
BNE(R31,...,XP)
108
MUL(...)
104
ADD(...)



Problem 5. Consider execution of the following code sequence on our pipelined Beta processor:

```
ADDC(R31, 3, R0)
SUBC(R0, 1, R1)
MUL(R0, R1, R2)
XOR(R0, R2, R3)
ST(R3, 0x1000, R31)
```

A. What value gets stored into location 0x1000?

```
ADDC(R31, 3, R0)      | R0 = 3
SUBC(R0, 1, R1)       | R1 = 2, R0 bypassed from ALU
```

```
MUL(R0, R1, R2)       | R2 = 6, R0 bypassed from MEM, R1 bypassed from ALU
XOR(R0, R2, R3)       | R3 = 5, R0 bypassed from WB, R2 bypassed from ALU
ST(R3, 0x1000, R31)   | 5 is stored in location 0x1000, R3 bypassed from ALU
```

B. At what point during the execution of the above sequence is data bypassed from the Memory stage to the ASEL or BSEL input multiplexors?

When the MUL is in the RF stage.

C. The above sequence is executed on a faulty Beta, whose only problem is that data bypassed from the WB stage is always presented to the ASEL and BSEL multiplexors as zero. What value will be written into memory location 0x1000 using this faulty Beta?

```
ADDC(R31, 3, R0)      | R0 = 3
SUBC(R0, 1, R1)       | R1 = 2, R0 bypassed from ALU
MUL(R0, R1, R2)       | R2 = 6, R0 bypassed from MEM, R1 bypassed from ALU
XOR(R0, R2, R3)       | R3 = 6, R0 bypassed from WB (as 0), R2 bypassed from ALU
ST(R3, 0x1000, R31)   | 6 is stored in location 0x1000, R3 bypassed from ALU
```

D. Now the same sequence is executed on a different faulty Beta. In this case, all data read from the register file on either port reads as zero. What value will the above sequence write into memory location 0x1000 using this processor?

5! As shown in the answer to part (A) no accesses are made to register file when fetching register contents, bypass paths are always used.

Problem 6. Flaky Betas Inc.'s purchasing agent, Penny Pincher, has acquired a large number of 5-stage pipelined Betas with full bypass and 1 annulled branch delay slot (the FB3). These processors have a single flaw: the connection of the PC inputs to the WDSEL multiplexor is defective. Penny is proud of the deal she made, but the FBI software team points out that procedure calls are broken since the write to the LP register uses the broken path.

After a moment's thought, Penny proposes that a call to a nearby procedure f, rather than generating a BR(f,LP)be compiled as:

```
LDR(.+8,LP)
BR(f,r31)
LONG(.+4)
```

A. Which of the following is the best statement about Penny's scheme?
   A. it works
   B. it will work only if the return sequence from a procedure is modified to add 4 to the value of LP before executing a JMP(LP) to return to the caller
   C. doesn't work since the LDR instruction is also broken by the flaw in the FB3's data path
   D. only works if the program is placed in the bottom 32767 words of main memory.
   E. doesn't work

(A) The LDR loads the address of the instruction following the LONG() into LP, so when the procedure returns, execution resumes at the correct point

B. Whether Penny's proposed scheme works or not, the software team doesn't like it and demands another solution. Penny remembers that all of the bypass paths in the Beta design are still operational except for the one at the last (write back) stage.

On a procedure call, during the cycle in which the first instruction of the called procedure is in the RF pipeline stage, where in the 5-stage Beta pipeline is the return address?

In the memory pipeline stage. Instruction sequence is BR, annulled instruction, first instruction of procedure, i.e., the BR is two stages further along the pipeline.

C. Assuming that all procedures are compiled with a standard entry and exit code sequence and that we don't need to worry about interrupts, does the fact that the bypass paths still work help Penny generate another solution?

Yes! But the entry sequence will have to be modified by adding ADDC(LP,0,LP) as the first instruction to keep from loosing the value because of the FB3 flaw. Adding this instruction bypasses the value of LP from the M stage (see answer to (B) above) just before it gets mashed by the broken path.

Problem 7. Pipelines-R-Us, a processor-design consulting firm located in the Valley, has submitted the following proposal to the 6.004 staff. They have noticed that the MEMORY stage of the five-stage pipelined Beta isn't used except during load and store operations. They propose omitting that stage entirely whenever the memory isn't used, as illustrated by the following table showing how an instruction travels through the various pipeline stages in succeeding cycles:

Instruction	*I*	*I+1*	*I+2*	*I+3*	*I+4*
LD/ST/etc.	IF	RF	ALU	MEM	WB
ADD/SUB/etc.	IF	RF	ALU	WB	

P-R-U reasons that instructions that leave out the MEM stage can complete a cycle earlier and thus most programs will run 20% faster!

In your answers below assume that both the original and the P-R-U pipelined implementations are fully bypassed.

A. Explain briefly to P-R-U why decreasing the latency of a single instruction does not necessarily have an impact on the throughput of the processor. Hint: consider how long it would take the original pipelined Beta to complete a sequence of 1000 ADDs. Then compare that with how long a P-R-U-modified Beta would take to complete the same sequence.

In the original pipelined Beta it will take 5 clock periods (T) to complete the first ADD and then 999T to complete the rest. Thus the total time to process 1000 ADDs will be 1004T. In the P-R-U modified Beta it will take 4T for the first ADD and 999T for the rest. So the total time for this "improved" Beta to complete 1000 ADDs will be 1003T, which is not much improvement over the regular pipelined Beta.

B. Consider a sequence of alternating LD and ADD instructions. Assuming that the LD instructions use different source and destination registers than the ADD instructions (i.e., there are no stalls introduced due to data dependencies), what is the instruction completion rate of the original, unmodified 5-stage Beta pipeline?

With no stalls, the original Beta completes alternating LDs and ADDs at one instruction per clock period T.

C. Now show how the same sequence of instructions will perform on a processor modified as P-R-U has suggested. Assume that the hardware will stall an instruction if it requires a pipeline stage that is currently being used by a previous instruction. For example, if two instructions both want to use the WB pipeline stage in the same cycle, the instruction that started later will be forced to wait a cycle. Draw a pipeline diagram showing where the stalls need to be introduced to prevent pipe stage conflicts.

If an instruction that uses all 5 stages (such as LD/ST/etc.) is executed right before an instruction that only uses 4 stages (ADD/SUB/etc.) then the second instruction will have to be stalled because the WB stage is in use and thus cannot be used at the same time by the second instruction. For example:

```
foo:    LONG( 0 )

        LD( foo, R0 )
        ADD( R1, R2, R3 )
```

To execute this sequence correctly the pipeline diagram must look like this:

Pipe Stage	t1	t2	t3	t4	t5	t6
IF	LD	ADD				
RF		LD	ADD			
ALU			LD	ADD		
MEM				LD	ADD (stalled)	
WB					LD	ADD

The stall occurs when the ADD and the LD attempt to use the WB stage at the same time, forcing the ADD instruction to remain in a wait stage during t5.

D. Did P-R-U's idea improve performance? Why or why not?

If all we ever executed were single 4-stage instructions such as one ADD, then P-R-U's idea would improve latency by 20%. In reality, however, we execute programs with many instructions. If these instructions are all 4-stage instructions, then, as shown in part (A) the performance improvement is insignificant. If these instructions are intermixed 4-stage and 5-stage instructions, then, as shown in part (C), there isn't any performance improvement.

Problem 8. Bargain Betas, Inc specializes in selling slightly defective Beta processors to budget-minded customers who are willing to program around the defects. BBI has acquired rights to the design of the Buba, a slightly defective version of the 5-stage pipelined Beta from lecture. The Buba differs only in its having no bypass logic or branch delay slot annulling.

You try running three little test sequences on the Buba, starting in each case with R1= -1, R2 = 1, R3 = 5, and R4 = -1:

```
S1:     ADD(R1, R2, R3)
        SUB(R2, R3, R4)
        CMPLT(R3, R4, R5)

S2:     ADD(R1, R2, R3)
        NOP
        SUB(R2, R3, R4)
        NOP
        CMPLT(R3, R4, R5)

S3:     ADD(R1, R2, R3)
        NOP
        SUB(R2, R3, R4)
        CMPLT(R3, R4, R5)
```

A. For each of the above sequences, give the value to be found in R5 (i) after execution on a working Beta and (ii) after execution on a Buba. Explain your answers.

In the Buba, a 5-stage pipeline without bypass logic, the result of an instruction will not be available to be read (in the Register File stage) until 4 clock cyles later. So for instance, if a certain instruction writes a value to R0, that value cannot be used by the next 3 instructions (they will all be reading the previous value of R0). The fourth instruction (following the original instruction) is the first that will be able to read the new value of R0. So, for instance, consider the following code:

```
ADDC(R31, 10, R0)
SUBC(R0, 5, R1)
ANDC(R0, 6, R2)
ORC(R0, 7, R3)
CMPLTC(R0, 11, R4)
```

Pipe Stage	i	i+1	i+2	i+3	i=4	i+5
IF	ADDC	SUBC	ANDC	ORC	CMPLTC	
RF		ADDC	SUBC	ANDC	ORC	CMPLTC
ALU			ADDC	SUBC	ANDC	ORC
MEM				ADDC	SUBC	ANDC
WB					ADDC	SUBC

The CMPLTC will be the first instruction to fetch the new value of R0. All the preceding instructions will be using the previous value(s) of R0. The ADDC instruction is in the Write Back stage while ORC is in the Register File stage-so the new R0 is not written back in time for the ORC to read it.

For the working Beta, S1, S2, and S3 all compute the same results. Initially: Reg[R1] = -1, Reg[R2] = 1, Reg[R3] = 5, Reg[R4] = -1

```
ADD( R1, R2, R3 )          Reg[ R3 ] = Reg[ R1 ] + Reg[ R2 ] = (-1) + 1 = 0
SUB( R2, R3, R4 )          Reg[ R4 ] = Reg[ R2 ] - Reg[ R3 ] = 1 - 0 = 1
CMPLT( R3, R4, R5 )        Reg[ R5 ] = (Reg[ R3 ] < Reg[ R4 ]) = (0 < 1) = 1
```

so **Reg[R5] = 1** for all three cases.

For the Buba (italics denote cases in which the Buba is different from a working Beta, in which the most recently calculated result is not being used):

```
S1:
ADD( R1, R2, R3 )          Reg[ R3 ] = Reg[ R1 ] + Reg[ R2 ] = (-1) + 1 = 0
   new value of Reg[R3] not available yet
SUB( R2, R3, R4 )          Reg[ R4 ] = Reg[ R2 ] - Reg[ R3 ] = 1 - 5 = -4
   new values of Reg[ R3 ] and Reg[ R4 ] not available yet
CMPLT( R3, R4, R5 )        Reg[ R5 ] = (Reg[ R3 ] < Reg[ R4 ]) = (5 < -1) = 0
   Reg[ R5 ] = 0

S2:
ADD( R1, R2, R3 )          Reg[ R3 ] = Reg[ R1 ] + Reg[ R2 ] = (-1) + 1 = 0
NOP
   new value of Reg[ R3 ] not available yet
SUB( R2, R3, R4 )          Reg[ R4 ] = Reg[ R2 ] - Reg[ R3 ] = 1 - 5 = -4
NOP
```

```
   new value of Reg[ R4 ] not available yet (but Reg[ R3 ] is available)
CMPLT( R3, R4, R5 )      Reg[ R5 ] = (Reg[ R3 ] < Reg[ R4 ]) = (0 < -1) = 0
   Reg[ R5 ] = 0

S3:
ADD( R1, R2, R3 )        Reg[ R3 ] = Reg[ R1 ] + Reg[ R2 ] = (-1) + 1 = 0
NOP
   new value of Reg[ R3 ] not available yet
SUB( R2, R3, R4 )        Reg[ R4 ] = Reg[ R2 ] - Reg[ R3 ] = 1 - 5 = -4
   new values of Reg[ R3 ] and Reg[ R4 ] not available yet
CMPLT( R3, R4, R5 )       Reg[ R5 ] = (Reg[ R3 ] < Reg[ R4 ]) = (5 < -1) = 0
   Reg[ R5 ] = 0
```

B. Describe how to add minimal bypass logic to the Buba so that the correct value will be left in R5 after the completion of sequence S2.

For S2 to function correctly, we need to add bypass logic so that the result of the ADD will be available for the SUB instruction. The ADD is 2 instructions ahead of the SUB, so the ADD will be in the MEM stage while the SUB is in the RF stage. So we need to add a bypass path from the ALU output in the MEM stage which goes back to the B operand in the RF stage. We also need the result of the SUB instruction for the CMPLE, but this can be accomplished with the same logic.

C. Describe what bypass paths are necessary to get the correct results in all three cases.

We have already discussed what is needed for S2. For S1:

- Result of ADD needs to be passed to the following SUB instruction; bypass required: from ALU output in ALU stage to B operand in RF stage
- Result of SUB needs to be passed to the following CMPLT instruction; bypass required: from ALU output in ALU stage to B operand in RF stage.
- Result of ADD needs to be passed to the CMPLT instruction (2 cycles later); bypass required: from ALU output in MEM stage to A operand in RF stage.

For S3:

- Result of ADD needs to be passed to the SUB instruction, following two cycles later; bypass required: from ALU output in MEM stage to B operand in RF stage
- Result of SUB needs to be passed to the following CMPLT instruction; bypass required: from ALU output in ALU stage to B operand in RF stage
- Result of ADD needs to be passed to the CMPLT instruction, three cycles later; bypass required: from ALU output in WB stage to A operand in RF stage.

In total, there are 4 bypass paths required:

ALU output in ALU stage to B operand in RF stage
ALU output in MEM stage to A operand in RF stage
ALU output in MEM stage to B operand in RF stage
ALU output in WB stage to A operand in RF stage

Note: we will also need muxes to select between the possible operands for A and B

D. Add the minimal number of NOPs necessary to the following instructions to make it produce identical results on the Buba and a normal Beta:

```
ADD(R3, R4, R5)
SUB(R5, R6, R7)
ADD(R1, R2, R3)
MUL(R7, R1, R2)
ADD(R4, R3, R5)
CMPLE(R7, R8, R9)
DIV(R7, R8, R10)
BEQ(R5, done)
ADDC(R1, 1, R5)

ADD( R3, R4, R5 )
NOP
NOP
NOP                     | Reg[R5] has not yet been updated
SUB( R5, R6, R7 )
ADD( R1, R2, R3 )
NOP
NOP                     | Reg[R7] has not yet been updated
MUL( R7, R1, R2 )
ADD( R4, R3, R5 )
CMPLE( R7, R8, R9 )
DIV( R7, R8, R10 )
NOP                     | Reg[R5] has not yet been updated
BEQ( R5, done )
NOP
ADDC( R1, 1, R5 )
```

The NOP after the BEQ instruction is necessary so that ADDC will only be executed if the branch is not taken.

Problem 9. This problem concerns the effect of external interrupts on the 5-stage pipelined Beta with bypass paths and 1 branch delay slot with annulment (i.e., the instruction in the delay slot is not executed). Recall that if an external interrupt arrives in cycle I, then the address of the interrupt handler, XAdr, is loaded into the PC at the end of cycle I and that the instruction that occupied the IF stage during cycle I gets replaced with BNE(R31,XAdr,XP). Assume that these are the first lines of the interrupt handler:

```
XAdr:   ADDC(SP,4,SP)
        ST(R0,-4,SP)
        ...
```

First, consider this code fragment:

```
        . = 0x1234
start:  CMPLTC(R1,0,R2)
        SUB(R3,R2,R3)
        XOR(R0,R3,R0)
        MUL(R1,R2,R3)
        SHLC(R1,2,R4)
```

A. Complete the following pipeline diagram for normal execution of those instructions (i.e., no interrupts are asserted).

Pipe stage	*t1*	*t2*	*t3*	*t4*	*t5*	*t6*
IF	CMPLTC	SUB				
RF						
ALU						
MEM						
WB						

Pipe Stage	t1	t2	t3	t4	t5	t6
IF	CMPLTC	SUB	XOR	MUL	SHLC	
RF		CMPLTC	SUB	XOR	MUL	SHLC
ALU			CMPLTC	SUB	XOR	MUL
MEM				CMPLTC	SUB	XOR
WB					CMPLTC	SUB

B. Complete the following pipeline diagram assuming that an interrupt arrives in cycle t2. What value is saved in XP as the result of the interrupt? Where should the interrupt handler return to when it finishes? Why doesn't it just return to the instruction whose address is saved in XP?

Pipe stage	*t1*	*t2*	*t3*	*t4*	*t5*	*t6*
IF	CMPLTC	SUB				
RF						
ALU						
MEM						
WB						

Pipe Stage	t1	t2	t3	t4	t5	t6
IF	CMPLTC	SUB	ADDC	ST		
RF		CMPLTC	BNE	ADDC	ST	
ALU			CMPLTC	BNE	ADDC	ST
MEM				CMPLTC	BNE	ADDC
WB					CMPLTC	BNE

The interrupt causes the address 0x123C to be stored in XP. When the interrupt handler is done it should return to the SUB instruction at 0x1238. If it would return to the address in the XP, then the SUB instruction would never get executed, because it was not executed before the interrupt handler.

C. Now consider what happens when we include a branch in the instruction sequence:

```
skip:   BR(NEXT)
        CMPLTC(R1,0,R2)
```

```
        ADD(R3,R2,R3)

next:   XOR(R0,R3,R0)
        MUL(R1,R2,R3)
        SHLC(R1,2,R4)
```

Complete the diagram for normal execution of the instructions starting at skip.

Pipe stage	*t1*	*t2*	*t3*	*t4*	*t5*	*t6*
IF	BR	CMPLTC				
RF						
ALU						
MEM						
WB						

Pipe Stage	t1	t2	t3	t4	t5	t6
IF	BR	CMPLTC	XOR	MUL	SHLC	
RF		BR	NOP	XOR	MUL	SHLC
ALU			BR	NOP	XOR	MUL
MEM				BR	NOP	XOR
WB					BR	NOP

D. Complete the diagram assuming that an interrupt arrives in cycle t2. To what instruction will the handler return when it is finished? Why is this a problem?

Pipe stage	*t1*	*t2*	*t3*	*t4*	*t5*	*t6*
IF	BR	CMPLTC				
RF						
ALU						
MEM						
WB						

Pipe Stage	t1	t2	t3	t4	t5	t6
IF	BR	CMPLTC	ADDC	ST		
RF		BR	BNE	ADDC	ST	
ALU			BR	BNE	ADDC	ST
MEM				BR	BNE	ADDC
WB					BR	BNE

After the interrupt handler is finished, it will return to the CMPLTC instruction. That clearly is not the correct behavior

because we want the branch to be taken and CMPLTC to be annulled.

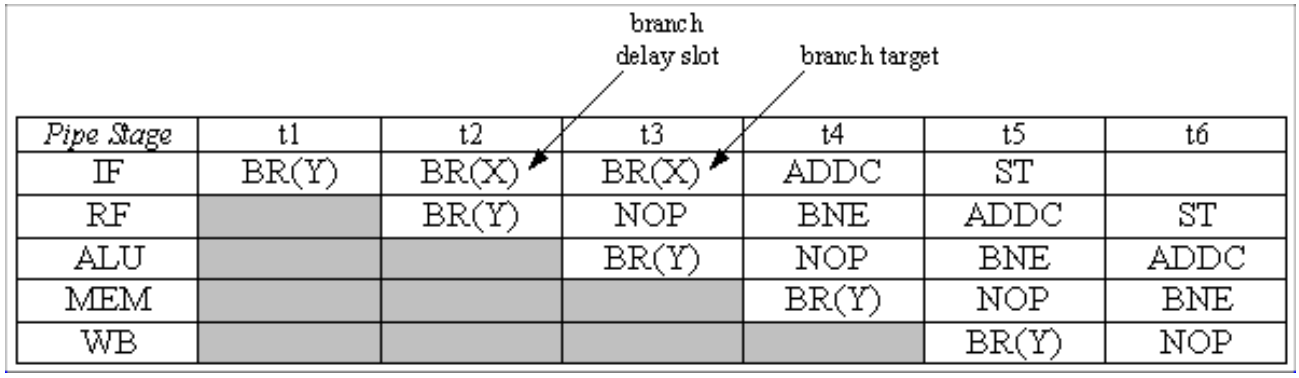
E. Normally interrupts are handled on the cycle on which they arrive, i.e., the instruction in the IF stage is discarded and a branch is forced to location Xadr. Suppose the hardware could be changed so that in some cases interrupts weren't handled on the cycle in which they arrived. In particular, suppose that interrupts were not allowed to occur when annulling an instruction in a branch delay slot. Explain how this solves the problem observed in part (D).

By not allowing interrupts when annulling the branch delay slot (cycle t2), then the value of PC+4 that gets stored in XP when the interrupt is handled (cycle t3) is the address of the second instruction following the branch, which in our example is the MUL instruction. When returning from the interrupt handler, the XP will be adjusted to the address of the XOR instruction.

F. Suppose interrupts are not allowed to occur when annulling an instruction in a branch delay slot. Will the following program create a loop that can't be interrupted?

```
X:      BR(Y)
Y:      BR(X)
```

No, the sequence is interruptible. If the interrupt arrives during t2, and we have disabled interrupts during branch delay slot annulment, then the interrupt will be serviced during t3.



Pipe Stage	t1	t2	t3	t4	t5	t6
IF	BR(Y)	BR(X)	BR(X)	ADDC	ST	
RF		BR(Y)	NOP	BNE	ADDC	ST
ALU			BR(Y)	NOP	BNE	ADDC
MEM				BR(Y)	NOP	BNE
WB					BR(Y)	NOP

The address stored in XP is the instruction following the BR(X), so when returning from the interrupt handler, then the XP is adjusted so it has the address of the BR(X). A similar argument could be made if the interrupt arrives while annulling the branch delay slot of the BR(X) instruction.