605.611 - Foundations of Computer Architecture

Assignment 07 - MIPS Datapath

Sabbir Ahmed

March 17, 2021

4.1 Consider the following instruction:

Instruction: `AND Rd, Rs, Rt`

Interpretation: `Reg[Rd] = Reg[Rs] AND Reg[Rt]`

4.1.1 What are the values of control signals generated by the control in Figure 4.2 for the above instruction?

Answer:

RegDst	Branch	MemToReg	ALUOp	MemWrite	ALUSrc	RegWrite
1	0	0	10 (AND)	0	0	1

4.1.2 Which resources (blocks) perform a useful function for this instruction?

Answer: All of the resources but the branch adder unit and the write port of the registers.

4.1.3 Which resources (blocks) produce outputs, but their outputs are not used for this instruction? Which resources produce no outputs for this instruction?

Answer: Write port of the registers and the branch adder do not use their outputs for this instruction. And all the resources produce outputs for this instruction.

4.6 When silicon chips are fabricated, defects in materials (e.g., silicon) and manufacturing errors can result in defective circuits. A very common defect is for one wire to affect the signal in another. This is called a cross-talk fault. A special class of cross-talk faults is when a signal is connected to a wire that has a constant logical value (e.g., a power supply wire). In this case we have a stuck-at-0 or a stuck- at-1 fault, and the affected signal always has a logical value of 0 or 1, respectively. The following problems refer to bit 0 of the Write Register input on the register file in Figure 4.24.

4.6.1 Let us assume that processor testing is done by filling the PC, registers, and data and instruction memories with some values (you can choose which values), letting a single instruction execute, then reading the PC, memories, and registers. These values are then examined to determine if a particular fault is present. Can you design a test (values for PC, memories, and registers) that would determine if there is a stuck-at-0 fault on this signal?

Answer: If the processor has a stuck-at-0 fault on the Write Register, then only the even registers are accessible by the instructions. We can test for this fault by attempting to write to an odd register and then checking its values.

We can test by loading the registers `$t2` with 0 and `$t3` with 1 and then executing:

```
ADD $t3, $t2, $t2
```

If this instruction was executed successfully, `$t3` would have the value of 0.

4.6.2 Repeat 4.6.1 for a stuck-at-1 fault. Can you use a single test for both stuck-at-0 and stuck-at-1? If yes, explain how; if no, explain why not.

Answer: Analogous to the stuck-at-o fault, if the processor has a stuck-at-1 fault on the Write Register, then only the odd registers are accessible by the instructions. We can use the same test as earlier with the registers flipped.

We can test by loading the registers `$t3` with 0 and `$t2` with 1 and then executing:

```
ADD $t2, $t3, $t3
```

If this instruction was executed successfully, `$t2` would have the value of 0.

The fault tests were done by assigning specific values to registers. Since there are no ways to assign both values to the registers within the same clock cycle, it is not possible to use the same test to detect both of the faults.

4.6.3 If we know that the processor has a stuck-at-1 fault on this signal, is the processor still usable? To be usable, we must be able to convert any program that executes on a normal MIPS processor into a program that works on this processor. You can assume that there is enough free instruction memory and data memory to let you make the program longer and store additional data. Hint: the processor is usable if every instruction "broken" by this fault can be replaced with a sequence

of "working" instructions that achieve the same effect.

Answer: If the processor has a stuck-at-1 fault on MemWrite, half of the registers become unavailable for use with any instructions. However, it is possible to execute instructions using the remaining odd registers.

4.6.4 Repeat 4.6.1, but now the fault to test for is whether the "MemRead" control signal becomes 0 if RegDst control signal is 0, no fault otherwise.

Answer: To test this fault, we need to execute an instruction that yields a value of 1 for "MemRead". Load instructions work in this case, as well as sets RegDst to 0. We can test by executing a load instruction with the immediate value of 0. We can check the value of the register loaded to and it should result in a different value since no memory was read. However, since the value read is random, it is also possible to read the immediate value that was loaded. Therefore, this test may have to be done several times with random immediate values.

4.6.5 Repeat 4.6.4, but now the fault to test for is whether the "Jump" control signal becomes 0 if RegDst control signal is 0, no fault otherwise.

Answer: To test this fault, we need to execute an instruction that yields a value of 1 for "Jump". Jump instructions work in this case. Since jump does not write to any register, RegDst is set to "don't care'. Since it is not possible to control the value of RegDst, there are no reliable tests to check this fault.

4.6.6 Add unit in the upper right, the ALU result bit 0 and result bit 31 are stuck at 0 & 1. Note that this will always work correctly or always fail.

Answer: If the bit 0 of the ALU result has a stuck-at-0 fault, and the 31st bit operated normally, the processor will execute as intended. This is because in the MIPS architecture, all the instructions are word-aligned with the last 2 bits always set to 0. If the 0-th bit has a stuck-at-1 fault, the program counter will end up non-aligned and would cascade the effects over the remaining set of instructions.

If the bit 31 of the ALU result has a stuck-at-0 fault, then the program counter would be accessing memory beyond the segments allocated for the data segments

and breach into kernel memory. It is possible for the processor to still function properly while accessing these memory segments as long as the kernel is not modified and deemed unusable. A stack-at-1 fault on the 31st bit results in no issues with the processor.

2 Program the microcode instruction, similar to the micro table we did in class, for an `R` instruction, `I` instruction, and `sll` instruction. Do the same for an `andi` instruction. What problems do you see for the instructions `sll` and `andi` instructions?

Instruction	RegDst	Branch	MemToReg	ALUOp	MemWrite	ALUSrc	RegWrite
`subu`	1	0	0	10 (SUB)	0	0	1
`subiu`	0	0	0	00 (??)	0	1	1
`sll`	1	0	0	10 (??)	0	0	1
`andi`	0	0	0	00 (??)	0	1	1

The `I` instructions (`subiu`) and `andi` do not have an ALUOp designated for them. Therefore, the microtable used for `R` instructions do not work with them. `sll`, even if an `R` instruction with designated ALUOp values, does not specify the values of `shamt` as required by the instruction.