Inf2C, Computer Systems: Tutorial 3, Week 7 Guide for tutors

Stratis Viglas

1. **Ripple Carry Adder**
   Assume that the propagation delay in each gate in a 32-bit ripple carry adder is 100ps. How fast can a 32-bit addition be performed? Assuming that the adder delay is the major limiting factor on the clock speed, how fast can we clock the processor?

   The longest path is from carry in at bit 0 to sum 31. For the full adders at bit positions 0-31, the carry propagates through 2 gates: one AND and one OR (cout = ab + ac + bc) For the last full adder the slowest path is to the sum output, not the carry output. The delay is 3 gates: inverter, AND, OR. ($s = \overline{a} \cdot \overline{b} \cdot c + \overline{a} \cdot b \cdot \overline{c} + a \cdot \overline{b}\overline{c} + a \cdot b \cdot c$) *So total 31 full-adders * 2 gates each + 3 gates for the last adder = 65 gate delays * 100ps = 6.5ns*

2. **Modulo-6 Counter**
   Design a synchronous Modulo-6 counter that will be able to count up to 5 clock positive edges. When it reaches 5, it resets to 0 and it starts the whole process again. In order to design this counter you can use D flip-flops and any two or three input gate you like.

   First note that we need to be able to count up to 6, which means that we need 3 bits to represent all the possible states of our FSM. The state machine can be seen in Figure 1. From this FSM we can derive the transition table and from that we have the circuit shown in Figure 2.

3. **Datapath**
   Discuss the steps in executing the `jal`, the `lw` and the `add` instruction in

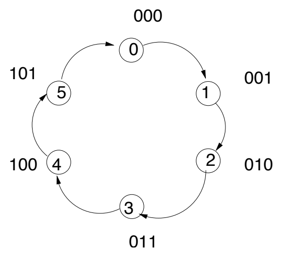

Figure 1: Modulo Counter FSM

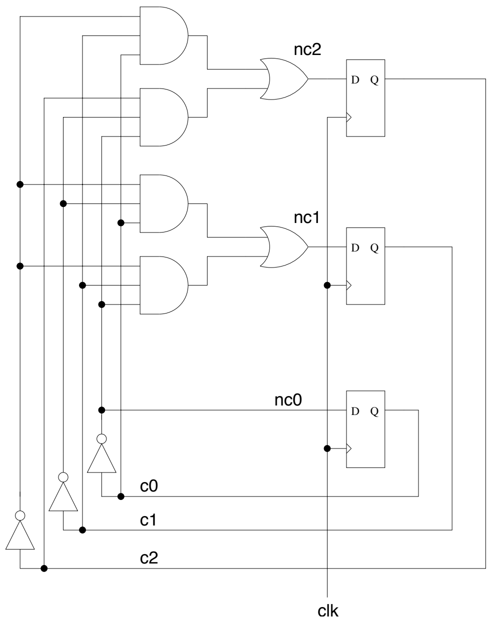

Figure 2: Modulo Counter Circuit

the multiclock datapath presented in Figure 3.

Now assume that an access to the register file takes 6.5ns, an access to the memory takes 100ns and the ALU delay is 6ns. Assume also that the instruction's execution stalls when it waits for a resource to produce a result. If we were to optimize this simple processor, which should be the component we should spend our main design efforts on, what could we do to improve it?

In the MIPS datapath all the instructions are executed in either 4 or 5 cycles depending on the operation required. It will be easier to take each instruction that needs to be analyzed, and break down the operations required for it's completion per cycle. The first two cycles are the same for all the instructions. During the first cycle, we fetch in the Instruction Register from the memory location indicated by the current PC, we also increase the PC.

In the second cycle, we decode the Instruction register and get which registers correspond to register A and register B, we also sign extend the 15 LSB to get, along with the PC the Immediate value.

JAL:
Since the JAL is a branch instruction in the third cycle we save the PC+4 held in ALUOut to register 31. This require a small modification to the datapath: we need to have 31 as an input to the multiplexer which provides

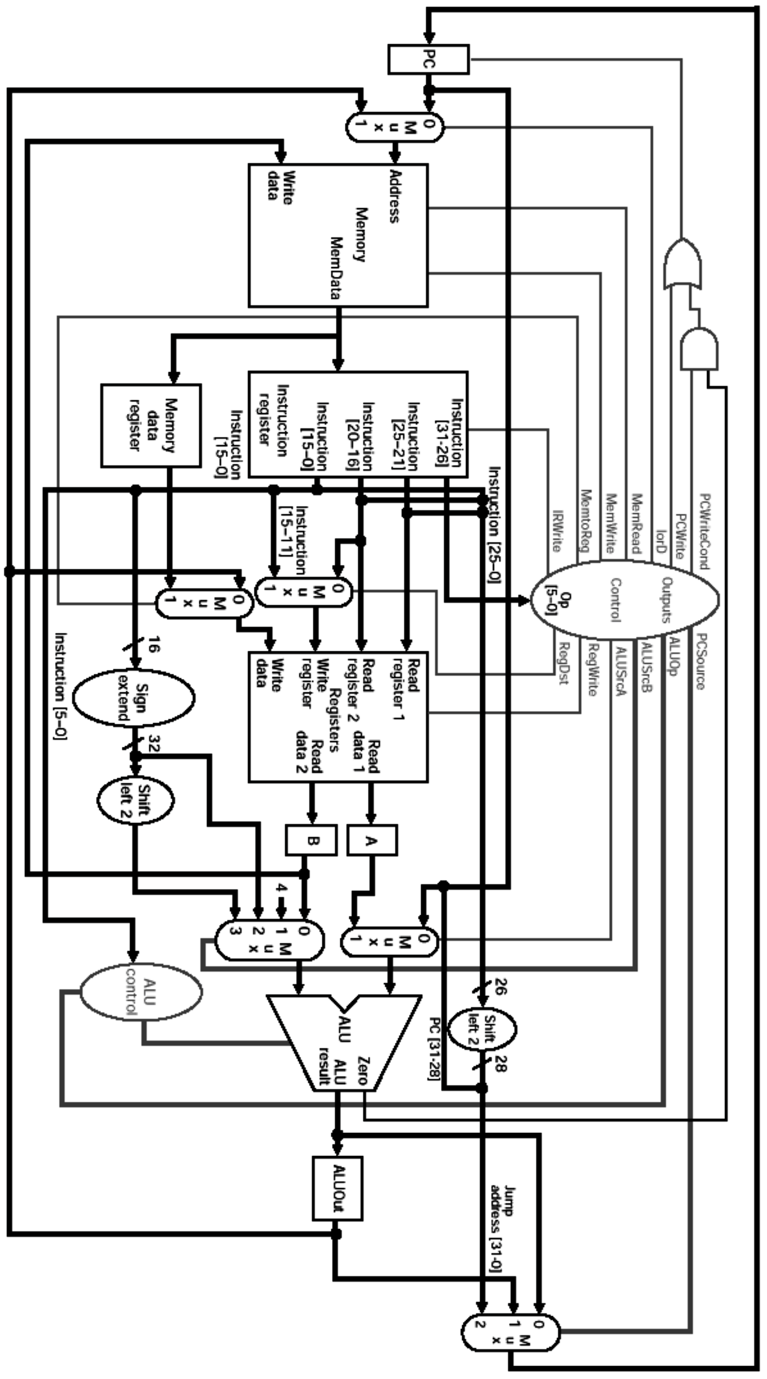

Figure 3: MIPS Datapath

the write register to the register file (mux control signal regDst. At the same time we compute the jump location using the shift left by 2 unit and concatenating with the upper bits of the PC (look above the ALU in the datapath).

LW:
The LW is a memory instruction, so in the third cycle we have to form the effective address, feeding the ALU with the A register and the sign extended Immediate value (cycle 2)

In the fourth cycle we load the Memory Data Register MDR with the memory location computed in the previous cycle.

In this final cycle, we load the register indicated by the bits 16 to 20 of the Instruction Register, with the value we have in the MDR.

ADD:
Finally we have this instruction is a register-register operation. So in the third cycle, we compute the addition using the ALU of the contents of register A and register B.

In the fourth cycle we load the register indicated by the bits 11to 15 of the instruction register with the computed value.

In the final section, note that the memory operation costs a lot more than the access to the register file and the ALU, therefore one should try to improve that. The obvious way to do it, is to use a cache which will provide a fast access for the vast majority of the memory operations. Caches have not been covered in lectures yet and will be discussed next week, so the point is to motivate optimising the memory system rather than looking at any specifics yet.

4. **Pipeline**
What is the advantage of having a pipelined processor? Make sure you recall what structural and data hazards are. Consider a five stage MIPS pipeline. Assume that the processor has one memory port. Track the state of the pipeline at each cycle while executing the following sequence of instructions.

```
add $t0, $t3, $t4
lw $t2, 4($t4)
add $t3, $t2, $t1
lw $t8, 3($t9)
add $t4, $t5, $t6
xor $t1, $t2, $t3
add $t7, $t3, $t4
```

Will adding a memory port change anything?

It will be useful to link to the previous part when talking about this, noting that most of the processor is idle at any given time. Hazards have been *covered in lectures and students should be familiar with them. Review if required.*

The instruction sequence will result in three instructions stalling in the pipeline.

Instruction 3 depends on the load in instruction 2, so a three cycle stall is introduced.

There is a potential structural hazard that arises because the IF stage of instruction 7 overlaps with the MEM stage of instruction 4. Instruction 6 (xor) needs to read t3, which is written in instruction 3 (add). This introduces one stall cycle. As a result the instruction fetch of instruction 7 (add) is delayed and no structural hazard occurs.

There is a further dependence between instructions 5 and 7.

Therefore in this particular code the structural hazard does not arise, however, it can happen with a different set of instructions.