Qianhui Wang U7070170
Linxi Li U7095375

Parameter	Delay (ps)
t_{pcq_PC}	40
t_{mem}	200
t_{dec}	70
t_{mux}	25
t_{RFread}	100
t_{ALU}	120
$t_{RFsetup}$	60

Fig. 1

BPIPE

Since there is not directly connection between ALU and MEM, and no relative addressing, so we don't need to separate EXE and MEM stages, that makes 4 stages: fetch, decode (reg_file read), execution and writeback (reg_file write). And we set 3 PPRs between these stages. As for the internal fragmentation, from data from Fig.1: IF stage: $t_{pcq\_PC}+t_{mem}=40+200=240ps$; ID stage: $t_{dec}+t_{RFread}+t_{RFsetup}=70+100+60=230ps$; EXE stage: Max{AND+OR+MUX, t_{mem}, t_{ALU}}+t_{mux}=225ps, which is a balance design for all stages. Data memory and instruction memory are combined, the memory is part of both EXE and IF stage. We put the "set high move low" circuit in EXE stage instead of ID stage. Because this reduces number of bits that need to store in ID/EX PPR without adding process time and makes the hardware simpler.

There are two structural hazards, which are the problem that different parts use the same resource simultaneously. First, the read and write of reg_file. In the traditional way, "first cycle writes second cycle read" waste one cycle to nop. To optimize we use the policy "first half write, second half read". We connect the raising clock with reg_file and memory (for write), and we negate the clock for PPRs, so they can read in falling edge. Second, since the combined memory, if the fetch and EXE stages need to read and write simultaneously, and the address for read and write are the same, the reading need to be pushed back and insert nop. PC don't need to push back. To detect structure hazard, check the store signal set high, and 1A port in RAM equal to 2A port. After the stall, read out from the same address and PC enable set low. To nop, just make all control unit signals zero. Actually, just set write enable and stall signal zero is enough, to prevent the content of reg_file and memory change.

DPIPE

The "true dependence" in "register-use" hazard and "load-use" hazard in MIPS. Since QuAC combined EXE and MEM, only 1 hazard here (less overlap in shorter pipeline), so we put the forwarding unit in EXE stage. We add forwarding path so reduce nop. If data dependence is detected in Rd, the circuit will pass the data from last instruction by forwarding path, else the data will be directly from reg_file. One difference from MIPS is, this already contain the "load-use" situation. Since "load-use" is mainly for memory, the ALU, memory and

“set high move low” circuit are parallel without any connection, so we use multiplexer to choose one of them. Since our design put ALU and MEM in the same stage, we can reuse the same hardware component to deal with both hazards.

CPIPE

We set the branch detection unit in ID stage. If branch detection unit detected branch taken, null the IF/ID register for two cycles, send nop to next cycle, jump to the correct instruction. We put the PC in reg_file, it can be written in by ALU or memory load. The conditional execution needs the flag of the most recent instruction in ALU, at the time that the ALU produces. But the reg_file have a half cycle delay, and the branch detection unit will miss the flag if store to the flag register first, so we need a multiplexer to select. We also store the flag from ALU into flag register in reg_file. Because this keeps the most recent flag, even if the last several instructions didn’t use ALU. The reason that we put PC register and flag register into reg_file is, the PC is general purpose read and write register, the flag register can be read and write by ALU, reuse the circuit in reg_file helps with saving hardware. Since the Rd register operational calculation, the new address will be detected until EXE stage, so we need to nop here. The forwarding path from EX/WB PPR to IF stage need two nop, so need to null IF/ID PPR twice.