Computer Architecture – Assignment #3

Pipeline Architecture

1. Introduction

In the pipelined architecture, there are several occasions when a necessary operation for an instruction cannot be performed because the requested resources or data are not available in the same pipeline stage at the same clock cycle. These events are called hazards, and there are three different types.

1. Structural hazards: The requested hardware resource is not ready to support the demanding instructions at the exact moment.
2. Data hazards: The result data of preceding instructions are going to be used for the coming instruction. But the result data are not delivered to the requesting pipeline stage at the exact moment.
3. Control hazards: The program is about to branch either way of two different flows. Then the following instructions from one of either direction should be not executed and should wait for the result of the branch instruction.

The following figure shows an example of similar pipelined architectures from the textbook.

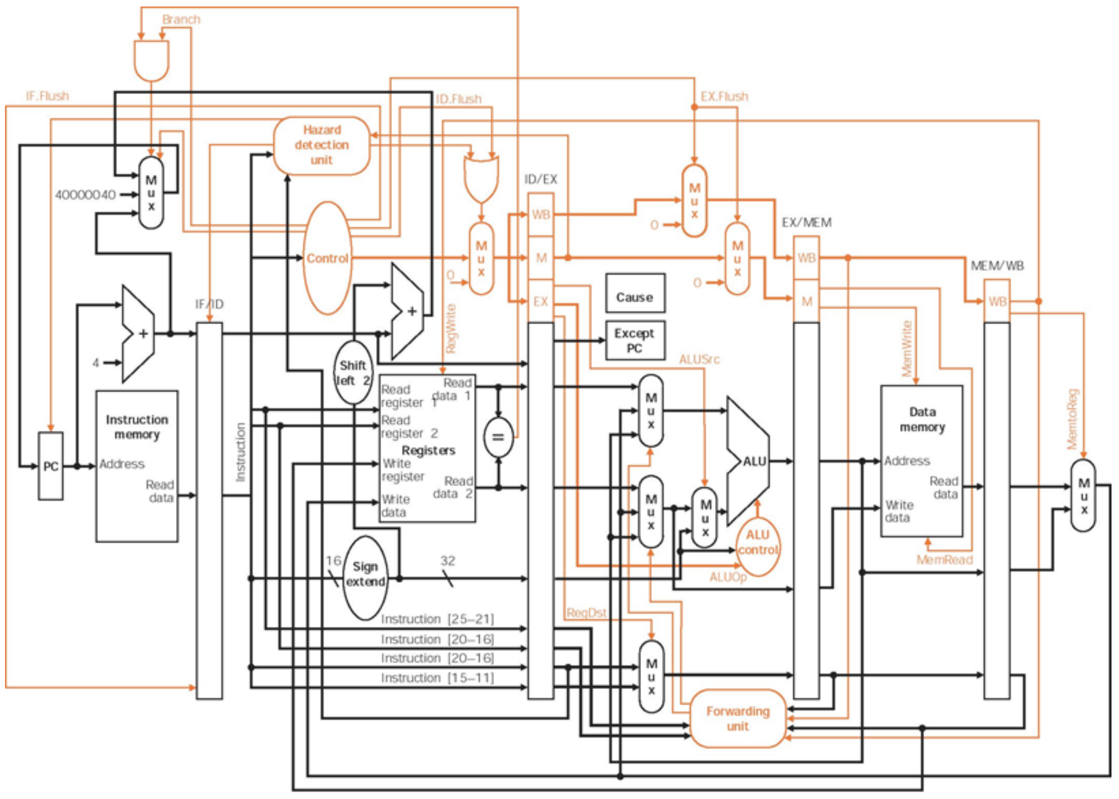


Figure 1 The example Pipeline CPU with hazards units

However, the pipelined processor **without** considering the hazards is given. In this assignment, modify the sorting program given by removing unnecessary NOPs and determine how to forward necessary data to avoid hazards. Simulate your code and discuss your results.

2. Assignment

There is no hazard if there is only one instruction in the pipeline. By placing 4 NOP instructions right after an instruction you can make only one useful instruction available in the pipeline. However, you can improve the performance by removing the NOP instructions if there is no dependency between the useful instructions.

2.1 Program Analysis and Simulation

- With the provided assembly code, find all Dependencies and draw them on the code as arrows. Draw dependencies on separate pages for each problem.
  1. Find all data dependencies and show them on the assembly code. On each arrow, indicate how many stalls are necessary.
  2. Find all control dependencies and show them on the assembly code as arrows. On each arrow, indicate how many stalls are necessary.
  3. Some stalls due to data dependencies can be resolved by data forwarding to ALU. Mark the only dependencies can be resolved by the ALU forwarding and give how many stalls can be reduced.

- Simulate the provided sort code and analyze results.
  1. Remove NOP instructions as many as possible without code rescheduling (Explain reasons).
  2. Remove more NOP instructions by inserting forward control signals.
  * There is no correct answer. Just minimize total cycles

*** Since branch instructions are executed in the decoding stage, ALU forwarding does not resolve data dependencies to branch instructions. The simulator does not have register file forwarding, either.**

2.2 Pipelined Processor Modules

Table 1 - Modules

Instance Name	Description	Instance Name	Description
i_clk	Clock	**i_rst_n**	Reset signal
PP_CPU	Top module. Pipeline Signals are shown here		
U13_RF	Register file	**U00_PC**	Program counter
U22_ALU	ALU	**U02_IM**	Instruction Memory
U23_MULT	Multiplier	**U31_DM**	Data Memory

Table 2 – Signals for M_TEXT_FWD.txt

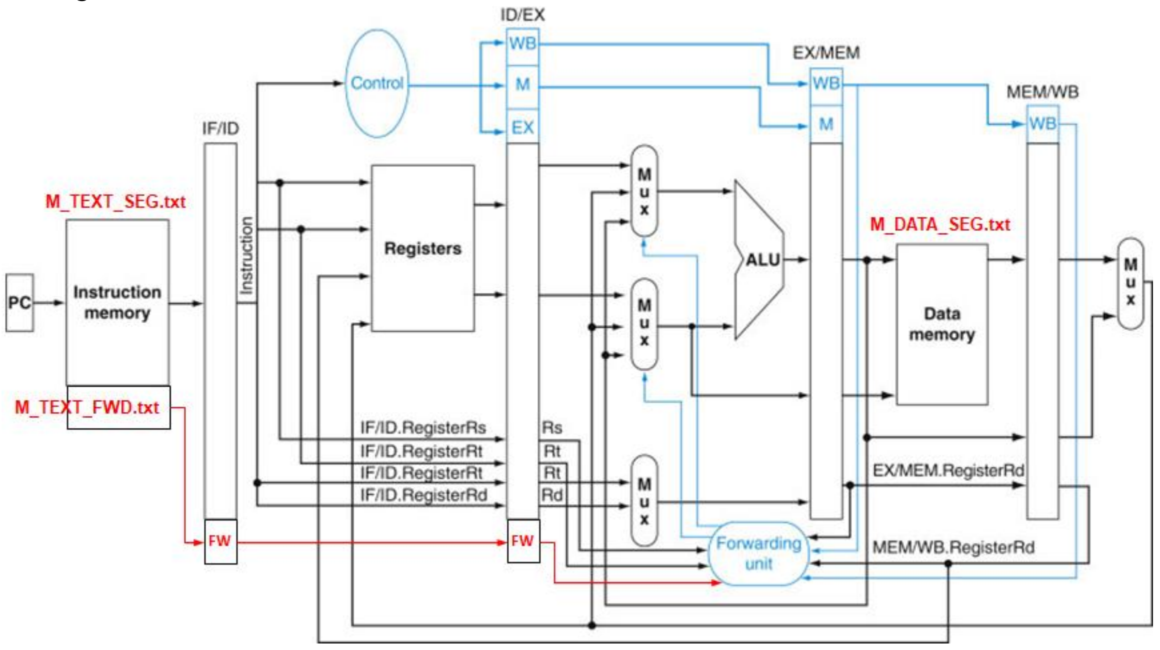
Signal Names	Descriptions
FWD_ALU_Ai	**00:** From Register file of ID stage to EX stage **01:** From ALU output of MM stage **10:** From Writeback data of WB stage
FWD_ALU_Bi	**00:** From Register file of ID stage to EX stage **01:** From ALU output of MM stage **10:** From Writeback data of WB stage
*** For both signals, 11 is not allowed.**	

2.3 Simulation Method

The sorting program code and data are given. Place the corresponding machine code in "**M_TEXT_SEG.txt**" after editing the program. The simulation finishes when it executes **BREAK** instruction or when it reaches the number of simulation counter to the number in "**tb_cycle_limit.txt**". The "**mem_dump.txt**" and "**reg_dump.txt**" show the data in 32bit word format which means first left-most byte has the highest address.

The signals in "**M_TEXT_FWD.txt**" forward data from appropriate pipeline registers to ALU inputs. The timing of forwarding is identical to the instruction in the "M_TEXT_SEG.txt", which means that the address of forwarding signal should match the addresses of instruction.

After simulating your code with **vvp** of Icarus Verilog tools; check the waveform carefully to see your data are ready for ALU or Branch on time. The signal names in top level represent their stage as: **w_STAGE_signalname**; for example, **w_EX_wb_cmd** means the writeback command signal in EX stage.



3. 결과 Report(2장 이상으로 작성)

- ■ 실험 내용
  - ● 프로젝트 이론 내용에 대한 설명 ex) Hazards 설명, Hazards를 피하기 위한 방법 (H/W-Forwarding, S/W-codeing)
- ■ 검증 전략, 분석 및 결과
  - ● 자신이 구현한 assembly code 설명
  - ● 2개의 시뮬레이션 결과 분석
  - ● 명령 수행에 걸린 총 cycle 수 등

1. 제시된 Radix Sort의 dependency 들을 코드가 적인 종이에 표시 및 설명

2. Radix Sort의 2가지 시뮬레이션 진행 및 설명

a. 기존 어셈블리코드에서 Forwarding 제어 없이 필요 없는 NOP만 제거한 시뮬레이션

b. 기존 어셈블리코드에서 Forward 제어신호를 추가하여 더 많은 NOP를 제거하여, 재구성된 어셈블리 코드 시뮬레이션

- Basic Block의 초기 instruction들은 Forwarding 방향이 두 가지로 나타날 수 있으므로 주의할 것. 동적으로 Hazard를 확인하는 실제 프로세서서는 문제가 없으나, 정적으로 Forwarding 방향을 지정하는 본 시뮬레이션에서는 문제가 생김.
- MIPS 어셈블리어와 machine code 사이 register operand의 순서가 다른 경우가 있음.

- ■ 문제점 및 고찰
  - ● 프로젝트 내용 전체 정리 및 고찰

- ■ Soft copy
  - ● **Due data: 2025/05/26(월요일) 23:59시까지. (딜레이 받지 않음.)**
  - ● 결과 Report를 작성한 파일을 압축하여 U-campus에 업로드.
  - ● 압축 파일명 양식: 학번_이름_Assignment#3.zip