Lauren Sherburne

Midterm 2: Quick Reference Guide

Hardware Resources Needed to Perform Basic Instructions

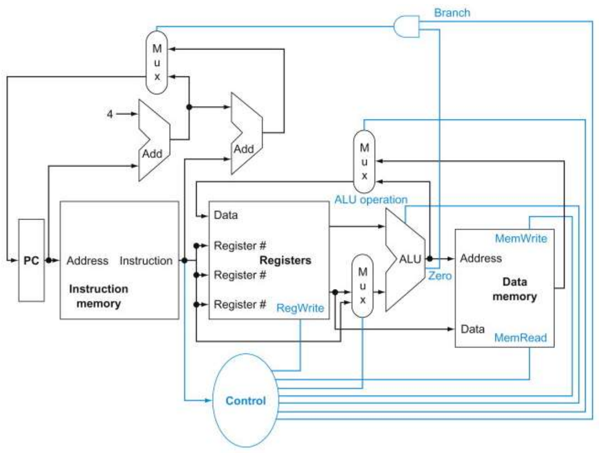

Figure 1 – Basic MIPS Implementation (Zybooks 3.1.2)

- Program Counter (PC): sequential/state element that is triggered by the clock; the register that carries the 32-bit address for the next instruction that will be executed
- Instruction Memory: sequential element that is only read and is never written thus treated as a combinational logic; not clock triggered; input address and output instruction are 32-bits
- Multiplexor (Mux): combinational element; the hardware version of an if-else statement; performs logical computations on two inputs to produce an output
- Register File: sequential element that uses the clock and regWrite to control writing to registers; the size of both registers and data are 32-bits
- Data Memory: sequential element that can be read, using lw instruction, or written to, using sw instruction; receives memWrite and memRead from control unit; write operation is triggered by clock; cannot simultaneously read and write
- Arithmetic-Logic Unit (ALU): combinational element that can perform addition, subtraction, set less than, bitwise and, bitwise or, return zero flag if equal (beq), and will flag overflow; receives ALUop from control and two 32-bit inputs; result is 32-bits
- Control Unit: using the instruction, control generates regDst, regWrite, ALUSrc, PCSrc, memRead, memWrite, memToReg
- Adder Block: combinational, data path element that takes in two inputs of 32-bits each and adds them to produce a 32-bit output; responsible for PC + 4 calculation
- Sign Extension Unit: combinational element used to change the size of the input to 32 bits by extending the first bit (sign bit)
- Clock: an electrical signal that switches on and off and is used to trigger hardware elements; rising edge starts each clock cycle

Actions Taken to Execute Basic MIPS Instructions within CPU (Central Processing Unit)

- Data Transfer: lw, sw
  - Using instruction memory, fetch the instruction
  - Prepare for the next instruction by incrementing PC (PC + 4)
  - Read the register file

    - Calculate the memory address: R[rs] + imm
    - Read from or write to data memory
    - Update the register file only if load instruction
- R-Type: add, sub, and, or, slt
    - Using instruction memory, fetch the instruction
    - Prepare for the next instruction by incrementing PC (PC + 4)
    - Read the register file
    - Operate on rs and rt registers
    - Update the register file
- Control Flow: beq
    - Using instruction memory, fetch the instruction
    - Prepare for the next instruction by incrementing PC (PC + 4)
    - Read the register file
    - Compare the rs and rt registers
    - Calculate the branch target address (BTA ={ 14{immediate[15]}, immediate, 2'b0 })

Purpose of the Register File, Data Memory, Instruction Memory within CPU

- Register File
    - Consists of 32 individual registers that are 32-bits each
    - Temporary and fast, but highly limited data storage
    - ALU only operates on registers, never on data memory
    - Accessed in phase two of the data path
    - Specialized Registers…
        - $0 – constant value: zero
        - $gp – global pointer
        - $sp – stack pointer
        - $fp – frame pointer
        - $ra – return address (after jump and link)
- Data Memory:
    - Only accessed by data transfer instructions (lw, sw) using specific addresses (base register and offset)
    - Contains billions of data elements and can hold data structures
    - Load word copies data from a memory into a register
- Instruction Memory:
    - Used in phase one to initiate an instruction

   - Uses the instruction to send signals to the control, which in turn dictates how each component in the data path will act
   - Sends signals to PC to increment PC properly

CPU Datapath – 5 Stages

- Phase 1: Instruction Fetch and PC + 4 (IF)
- Phase 2: Read the Register File (ID)
- Phase 3: Perform Operation (EX)
   - R-type: ALU performs given operation in order to compute the result
   - Data Transfer: the ALU performs an addition to compute the data address
   - Control Flow: first ALU compares the operands, second ALU computes the BTA
- Phase 4: Memory Access (MEM)
- Phase 5: Update the Register File (WB)

Results of MIPS Instruction at Each Stage through Datapath

- IF: retrieves the instruction from memory and copies it into the instruction register
- ID: control unit decodes and interprets the instruction and sends opcode to ALU, memRead and memWrite to data memory, regWrite to the register file, etc.
- EX: the ALU performs the instruction's computation(s) whether it is addition, subtraction, comparison, or determining the value of an address; the program counter is incremented in this phase
- MEM: data is either read from the long-term data memory into a register or it is loaded from a register into long-term memory
- WB: the result of the ALU computation or the data loaded from memory is written back to the specified register

Pipelining

- In short, pipelining overlaps the execution of multiple instructions which is possible since all instructions are the same length
- Once one instruction completes the IM (phase 1), it moves on to ID (phase 2), and the next instruction moves into
- Allows more instructions to be completed per unit of time

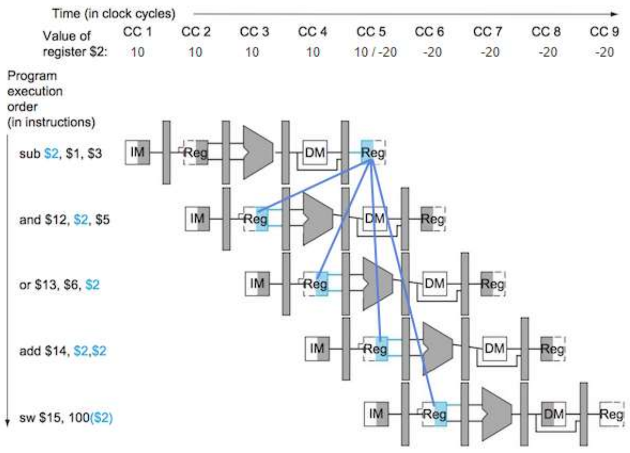

Figure 2 - Pipelining (Zybooks 3.7.1)

Impact of Pipelining on MIPS Instructions

- Pipelining speeds up the process of completing instructions since you do not have to go through all of the stages before returning to phase 1

Impact of MIPS Instructions on Pipelining

- Unlike the washing clothes pipelining analogy, MIPS instructions are not always uniform and they often create hazards that must be overcome.
- Figure 2 shows how future instructions may need to access registers that have not been updated yet, which causes incorrect results.

Data Hazards (Pipeline Hazards)

- Occurs when a future instruction cannot execute in the proper clock cycle since the necessary register has not been updated yet with the correct data and is not yet available
- The pipeline will only work properly if data hazards can be solved using one of two ways:
    - Forwarding – rather than waiting for the previous instruction to execute, the necessary data element is retrieved from internal buffers in the pipeline; also called bypassing
    - Stall – the pipeline "pauses" and waits one or more clock cycles before beginning the next instruction so that the necessary registers have been updated; also called bubble or nop

Control Hazards (Branch Hazards)

- Occurs when a branch is taken and the upcoming instruction that was fetched (phase 1) is not the one that is needed; the pipeline did not expect the flow of addresses that followed the branch
- The pipeline can use a stall (following any branch instruction) or it can flush the pipeline

When to Flush the Pipeline

- If the branch is taken, then the pipeline must be flushed and all of the instructions following the branch instruction are discarded
-

Computing CPU Performance

- The CPU Performance is the response time; it is the best parameter to measure a computer's performance; maximum performance = minimum response time
- Instruction Count (IC), Cycles Per Instruction (CPI), Clock Cycle (CC), and Cycle Length (T_C)
- the CPU Time for a Program = IC * CPI * T_C
- $\frac{performance_x}{performance_y} = \frac{CPUtime_y}{CPUtime_x} = how\ much\ faster\ x\ is\ than\ y$