Karan Vora (kv2154)
Computer Systems Architecture Assignment 5

Problem 1):

Solution 1.1):

Its a s-type instruction

opcode: 0100011
imm[4:0]: 10100
funct3: 011, sd store double word
rs1: 01101
rs2: 01100
imm[11:5]: 0000000

sd x12, 20(x13)

ALU control ALUop and Instruction [30, 14-12]
ALUop for sd instruction is 00
Instruction [30] = 0
Instruction [14:12] = 011

Solution 1.2):

PC + 4

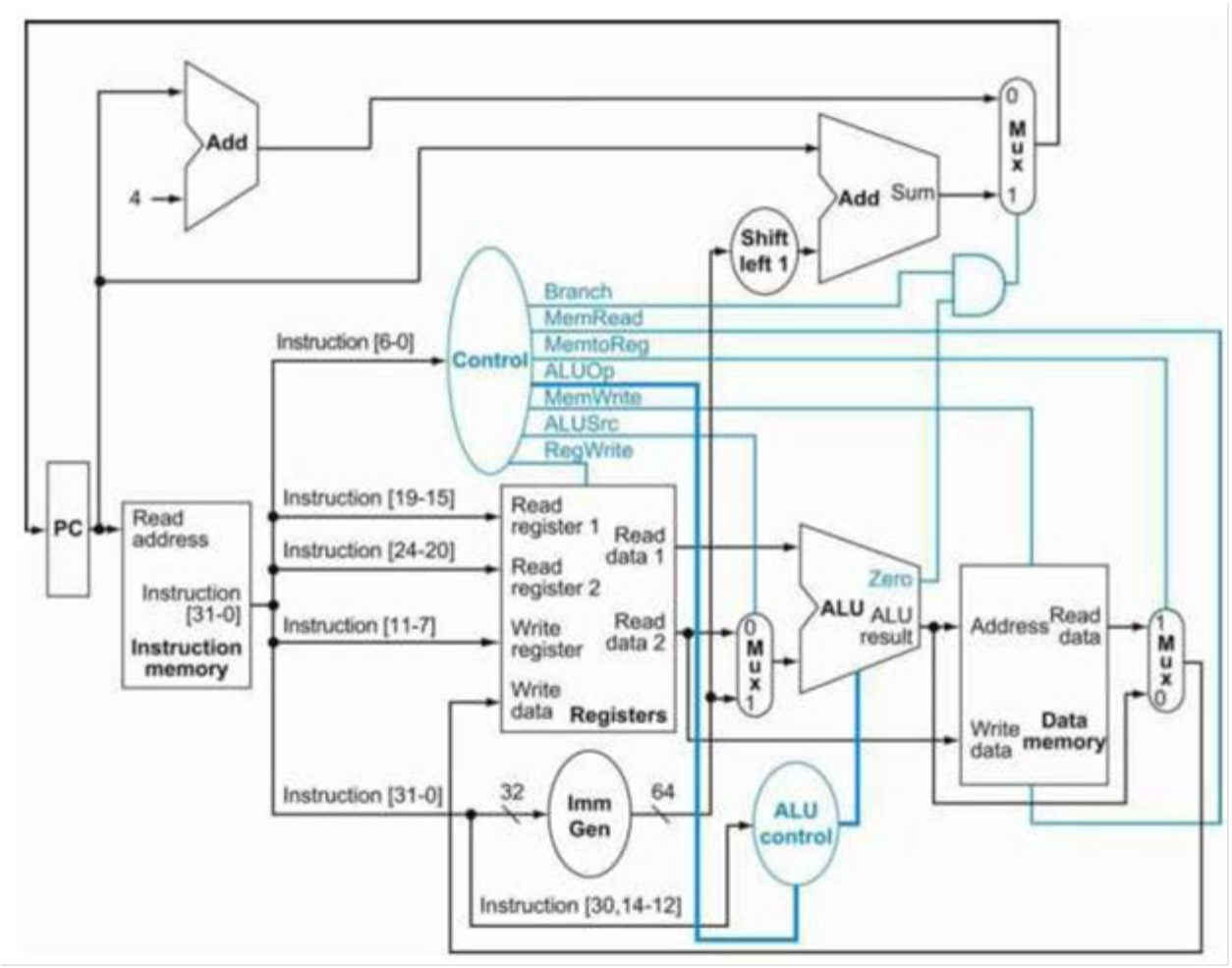

Image taken from textbook

Solution 1.3):

ALUSrc mux

Input 1: Reg[x12]
Input 1: 0x0000000000000014
Output: 0x0000000000000014

MemToReg mux

Input 0: Reg[x13] + 0x0000000000000014
Input 1: Undefined
Output: Undefined

PCSrc mux

Input 0: PC + 4
Input 1: 0x000000000000000A
Output: PC + 4

Solution 1.4):

a):
ALU: -3, 20
PC + 4: PC, 4
Branch: PC + 4, 20 x 4

b):
ALU: -32, -128
PC + 4: PC, 4
Branch: PC + 4, 2090 x 4

Solution 1.5):

a):
RR1: 3
RR2: 2
WR: Undefined
WD: Undefined
RegWrite: 0

b):
RR1: 4
RR2: 2
WR: 1
WD: 0
RegWrite: 1

==========

Problem 2):

Solution 2.1):

Latency for R-Type Instruction:
30 + 250 + 150 + 25 + 200 + 25 + 70 = 700 ps

Solution 2.2):

Latency of ld Instruction:
30 + 250 + 150 + 25 + 200 + 250 + 25 + 20 = 950 ps

Solution 2.3):

Latency of sd Instruction:
30 + 250 + 150 + 200 + 25 + 250 = 905 ps

Solution 2.4):

Latency for beq Instruction:
30 + 250 + 150 + 25 + 200 + 5 + 25 + 20 = 705 ps
Solution 2.5):

Latency for I-Type Instruction:
30 + 250 + 150 + 25 + 200 + 25 + 20 = 700 ps

Solution 2.6):

CPU Clock period: 950 ps

==========

Problem 3):

Solution 3.a1):

Using the latencies calculated in problem 2, the average time for our new sped up instruction set would be (0.52 * 700) + (0.25 * 950) + (0.11 * 905) + (0.12 * 705) = 785.65 ps. Speedup = 950/785.65 = 1.21 => ~ 21% speedup

Solution 3.b1):

Clock cycle time without improvement was: 950 ps
Adding the multiplier will increase the clock cycle by 300 ps
New clock cycle time = 905 + 300 = 1250 ps
Clock cycle after adding the multiplier with 5% reduced number of instruction = 0.95 * 1250 = 1187.5 ps

Solution 3.b2):

Speedup = 950/1187.5 = 0.8

Solution 3.b3):

The ALU time reduction = 5% = 0.05 * 950 ps = 47.5, but this new improvement adds a 300 ps penalty so the new improvement is not worth it.

Solution 3.c1):

Speedup achieved = 950 * (number of instructions) / 1250 * 0.95 * (number of instructions) = 0.8 => There is no real speed up, this will make the CPU slower

Solution 3.c2):

Register File cost is now 400. 12% reduction in load stores => (0.25 + 0.11) * 0.12 => 4.32% reduction in number of instructions. As compared to 950*n execution time, we now need 960 * 0.9568n execution time. speedup = 950/ (960 * 0.9568) = ~ 1.0342 => 3.42 % speedup. The relative increase in

cost = sum of new costs / sum of new costs = 4707/4507 => 4.4% increase in cost.

Solution 3.c3):

From what we know, a 4.4% cost increase results in a 3.42% increase in performance. We don't even know whether the increase in cost would scale linearly. Even if we
assume it does, the return on the increase in the cost by percentage is more than the%
increase in performance. If let's say a deadline is being missed by a few hours then this scaling may be worth looking into.

==========

Problem 4):

Solution 4.1):

The new clock cycle would eliminate the need for ALU which would have cost 750 ps

Solution 4.2):

Doubling of the ld/sd instructions means that the total number of instructions is now 1.36 * n. The new clock time is 750ps => the speedup would be 950 * n / (750 * 1.36 * n) = 0.931. there is no speedup, the machine has slowed down owing to the change in clock time and increase in instructions.

Solution 4.3):

In the new CPU, the number of loads and stores dominates how the CPU will perform.

Solution 4.4):

That is a highly subjective question. The newer CPU will do well when the number of load/stores are low and the older CPU will do better with high load stores is my guess.

==========

Problem 5):

Solution 5.a1):

We would need to add a new muxes to the datapaths for this implementation.

Solution 5.a2):

No functional blocks would require modification, just addition of data paths and muxes would be enough

Solution 5.a3):

We would need a path from the ALU output directly to the data memory and a path from read data to

the data memory's address input.

==========

Solution 6):

Solution 6.1):

Non pipelined would be the sum of all the stages = 1250ps
Pipelined would be the longest stage i.e ID stage = 350ps

Solution 6.2):

ld instruction would make use of all the stages so the latency would be the same for pipelined and non pipelined processor = 1250ps

Solution 6.3):

ld being the lengthiest in terms of time, splitting that would reduce the clock cycle time the most.

Solution 6.4):

Load % + Store % = 35%

Solution 6.5):

ALU/Logic % + Jump/Branch % = 65%

==========

Problem 7):

For a k stage pipeline, the first instruction doesn't enter the last stage i.e the WB stage will k cycles are done. After that the rest of the n-1 instructions will enter the WB stage one by one. Which gives us a minimum of k + n - 1 cycles.

==========

Problem 8):

Solution a):

Assume that x11 is initialized to 11 and x12 is initialized to 22. Suppose you executed the code below on a version of the pipeline from Section 4.5 that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). What would the final values of registers x13 and x14 be?
addi x11, x12, 5
add x13, x11, x12

addi x14, x11, 15

Since data hazards are not handled, x13 will contain 33 and x14 will contain 26

Solution b):

Assume that x11 is initialized to 11 and x12 is initialized to 22. Suppose you executed the code below on a version of the pipeline from Section 4.5 that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). What would the final values of register x15 be? Assume the register file is written at the beginning of the cycle and read at the end of a cycle. Therefore, an ID stage will return the results of a WB state occurring during the same cycle. See Section 4.7 and Figure 4.51 for details.
addix11, x12, 5
addx13, x11, x12
addix14, x11, 15
addx15, x11, x11

x15 will contain 54

Solution c):

Add NOP instructions to the code below so that it will run correctly on a pipeline that does not handle data hazards.

addix11, x12, 5
addx13, x11, x12
addix14, x11, 15
addx15, x13, x12

addix11, x12, 5
NOP
NOP
addx13, x11, x12
addix14, x11, 15
NOP
addx15, x13, x12