Tu Lam

CS 472 / Justin Goins

May 6th, 2021

Homework #3
(CPU)

1. *Derive the even-parity Hamming code that corresponds to the following data bits. Be sure to show your work and illustrate how each parity bit was determined.*

0b10111000101110

Answer: Below is the calculation of how each parity bit was determined:

Parity Bit	P1	P2	1	P4	0	1	1	P8	1	0	0	0	1	0	1	P16	1	1	0
P1	X		X		X		X		X		X		X		X		X		X
P2		X	X			X	X			X	X			X	X		X	X	
P4				X	X	X	X					X	X	X	X				
P8								X	X	X	X	X	X	X	X				
P16																X	X	X	X

Base off from the table chart, we get the parity bits to be:

P1 = (P1, 1, 0, 1, 1, 0, 1, 1, 1, 0) = 6 bits of 1s (even = **0**)

P2 = (P2, 1, 1, 1, 0, 0, 0, 1, 1, 1) = 6 bits of 1s (even = **0**)

P4 = (P4, 0, 1, 1, 0, 1, 0, 1) = 4 bits of 1s (even = **0**)

P8 = (P8, 1, 0, 0, 0, 1, 0, 1) = 3 bits of 1s (odd = **1**)

P16 = (P16, 1, 1, 0) = 2 bits of 1s (even = **0**)

After doing it, our Hamming code is:

0b0010011110001010110

2. *Suppose that you receive a Hamming code with the following contents. (containing data bits and parity bits):*

0b111011010010101

a) *How many bits of data (not including parity bits) were originally encoded?*

Answer: Looking at the Hamming code, there is a total of 15 bits of binary numbers. We know that parity bits are located at position 1, 2, 4, 8… so it looks like parity 1, 2, 4, and 8 can fit in this code (total of 4), leaving 15 - 4 = **11 bits of data were originally encoded.**

b) *Assuming that the transmitted Hamming code was using even parity, is it possible to determine the original data bits that were encoded? If so, derive the original data bits (show your work). If not, explain why.*

Answer: Giving the Hamming code below:

Bit	1	1	1	0	1	1	0	1	0	0	1	0	1	0	1
Parity	P1	P2		P4				P8							

We then could find out the parity bits of each to see if it is a correct parity bit (where even of 1s = 0, odd of 1s = 1):

P1 = (1, 1, 0, 0, 1, 1, 1) = 5 bits of 1s (parity is correct)
P2 = (1, 1, 0, 0, 1, 0, 1) = 4 bits of 1s (parity is incorrect)
P4 = (1, 1, 0, 0, 1, 0, 1) = 4 bits of 1s (parity is correct)
P8 = (0, 0, 1, 0, 1, 0, 1) = 3 bits of 1s (parity is correct)

From this we determine that position 2 is flipped, so make it even parity, we need to flip it to 0. So, the data will be **0b101011010010101** to be correct.

c) *Now assume that the transmitted Hamming code was using odd parity. Is it possible to determine the original data bits that were encoded? If so, derive the original data bits (show your work). If not, explain why.*

Answer: We then could find out the parity bits of each to see if it is a correct parity bit (Where even of 1s = 1, odd of 1s = 0):

P1 = (1, 1, 0, 0, 1, 1, 1) = 5 bits of 1s (parity is incorrect)
P2 = (1, 1, 0, 0, 1, 0, 1) = 4 bits of 1s (parity is correct)

P4 = (1, 1, 0, 0, 1, 0, 1) = 4 bits of 1s (parity is incorrect)
P8 = (0, 0, 1, 0, 1, 0, 1) = 3 bits of 1s (parity is incorrect)

From this we determine that parity bit 1, 4, and 8 is incorrect, so to determine the position, we add it up to get 1 + 4 + 8 = 13. position 13 is flipped, so make it odd parity, we need to flip it to 0. So, the data will be **0b111011010010001** to be correct.

3. *Imagine that you are considering two potential implementations of a virtual memory system (case "a" and case "b").*

a) *Assuming a memory page of 16K and a 32-bit virtual address space, calculate the amount of memory (in bytes) that is needed to store the full page table. Assume that each PTE requires 5 bytes. Show all calculations.*

Answer: We know that the virtual address has 32-bit, and the memory page is 16K (which is equivalent to 2^{14}). From this data, we can find the amount of entries store in the frame number by taking:

Memory Page (Virtual Page Number): $\mathbf{2^{14}}$ = 16,384 Bytes ≈ 16K
Entries: 2^{32} / 2^{14} = $\mathbf{2^{18}}$ **entries**

Then we can calculate the amount needed to store the full page table by taking: 2^{18} * 5 bytes of PTE = 1MB or **1,310,720 Bytes**.

b) *If the virtual address space is expanded to 48bits (and all other properties remain identical), how many bytes of memory are now required to store the full page table?*

Answer: We know that the virtual address has 48-bit, and the memory page is 16K (which is equivalent to 2^{14}). From this data, we can find the amount of entries store in the frame number by taking:

Memory Page (Virtual Page Number): $\mathbf{2^{14}}$ = 16,384 Bytes ≈ 16K

Entries: 2^{48} / 2^{14} = $\mathbf{2^{34}}$ **entries**

Then we can calculate the amount needed to store the full page table by taking: 2^{34} * 5 bytes of PTE = 85GB or **85,899,345,920 Bytes**.

4. *Consider Figure 4.10 in the textbook (the image is identical for the 5th edition and 6th edition).*

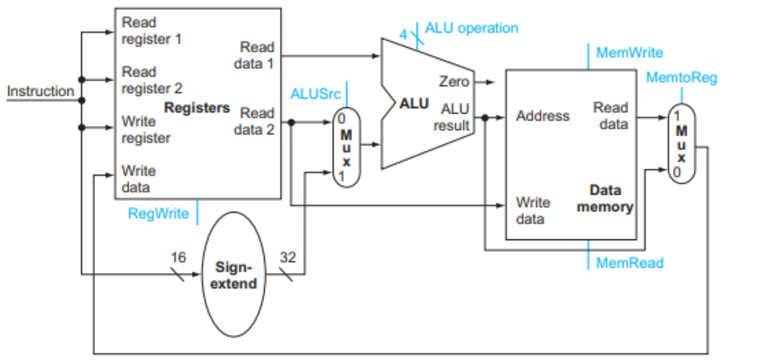

FIGURE 4.10 **The datapath for the memory instructions and the R-type instructions.** This example shows how a single datapath can be assembled from the pieces in Figures 4.7 and 4.8 by adding multiplexors. Two multiplexors are needed, as described in the example.

a) *Determine the control signals which will implement the following MIPS instruction:*

addi $15, $7, -36

Answer: From this we can determine the control signal with the table as the following:

RegWrite	ALUSrc	ALU Op	MemWrite	MemRead	MemtoReg
1	1	0010	0	X	0
Reg $15 is written by the instruction	For the sign-immediate value "-36"	For "addition" operation	No data write to mem	No reading content from data mem	Write back to register, never from mem to reg

b) *Similar to part A, determine the binary value of all control signals needed to implement the following MIPS instruction:*

lw $22, 76($10)

Answer: From this we can determine the control signal with the table as the following:

RegWrite	ALUSrc	ALU Op	MemWrite	MemRead	MemtoReg
1	1	0010	0	1	1
Reg $22 is written by the instruction	Using immediate value offset "76"	For "addition" operation	No data write to mem	Reading data content from mem	Output data is written from mem to reg

5. *In this question we will be working with a CPU that exhibits the following latencies for each stage of the datapath*:

IF	ID	EX	MEM	WB
250ps	270ps	190ps	330ps	225ps

For our particular program, the percentage of instructions (in each category) are as follows:

ALU/Logic	Load	Store	Jump/Branch
52%	18%	17%	13%

a) *What will be the clock period for a pipelined implementation?*

Answer: We see that the longest time in the datapath is at MEM with 330ps so that it will cover every other stage in datapath as pipeline like it when all stage is balance, so we have **330ps** for pipelined implementation so that it fit the clock cycle for every other stages in the datapath.

b) *What minimum clock period is expected for a non-pipelined design?*

Answer: Given the information, we can calculate the minimum clock period for non-pipelined is to add up the time it takes for each stage in the datapath and we get:

250ps + 270ps + 190ps + 330ps + 225ps = **1265 ps** of minimum clock period

c) *Suppose we want to improve the performance of our pipeline. We've been informed that it's possible to split a single stage of the pipeline into three stages, but there's a caveat: each of the three newly created stages will exhibit 40% of the original latency (not 33.3%). Which stage would be the best to split, and what latency would be exhibited by the newly pipelined CPU? I.e. how much time would this new CPU implementation require to fully execute an instruction?*

Answer: The stage that we would be best to split is the longest stage which is the **MEM stage with 330ps**. And the new latency that would exhibit is 7 (4 original + 3 new stages) * 270ps (second longest clock period) = **1890ps**.

d) *Ignoring the effects of stalls and hazards, what is the utilization of the data memory? Express your answer as the percentage of clock cycles when the data memory is actively being used.*

Answer: Since the data memory only utilize the load and store instruction in MIPS, then the percentage of clock cycle of data memory is 18% + 17% = **35% of utilization of data memory.**

e) *Ignoring the effects of stalls and hazards, what is the utilization of the write port on the register bank? Note: I'm asking specifically asking about the write port (controlled by the RegWrite signal in Figure 4.10). Express your answer as the percentage of clock cycles when the register write circuitry is being used to update a register.*

Answer: For the write port, it looks like it may utilize the ALU and the load instruction in MIPS, then the percentage of clock cycle of write port is 52% + 18% = **70% of utilization of the write port.**