CS 572: Computer Architecture **(Due: May 9th, 2021)**

Submission Homework 3

Instructor: Prof. Justin Goins *Name:* Rashmi Jadhav, *Student ID:* 934-069-574

1. The even-parity Hamming code corresponding to the data `0b 1011 1000 1011 10` is `0b 0110 0111 1000 1010 110` and the steps for the same are described below:

Placing parity bits and data bits for hamming code calculation

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11	P5	D12	D13	D14
_	_	**1**	_	**0**	**1**	**1**	_	**1**	**0**	**0**	**0**	**1**	**0**	**1**	_	**1**	**1**	**0**

Parity Bit 1 (Skip intevals of length 1)

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11	P5	D12	D13	D14
_		**1**		**0**		**1**		**1**		**0**		**1**		**1**		**1**		**0**

#1's = 6, thus P1 = 0

Parity Bit 2 (Skip intevals of length 2)

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11	P5	D12	D13	D14
	_	**1**			**1**	**1**			**0**	**0**			**0**	**1**			**1**	**0**

#1's = 5, thus P2 = 1

Parity Bit 3 (Skip intevals of length 4)

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11	P5	D12	D13	D14
			_	**0**	**1**	**1**					**0**	**1**	**0**	**1**				

#1's = 4, thus P3 = 0

Parity Bit 4 (Skip intevals of length 8)

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11	P5	D12	D13	D14
							_	**1**	**0**	**0**	**0**	**1**	**0**	**1**				

#1's = 3, thus P4 = 1

Parity Bit 5 (Skip intevals of length 16)

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11	P5	D12	D13	D14
															_	**1**	**1**	**0**

#1's = 2, thus P5 = 0

Final even-parity Hamming code

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11	P5	D12	D13	D14
0	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**

2. (a) Denoting the parity bits(at the positions of powers of two) and data bits in the given Hamming code:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11
1	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**

Thus, **11 bits of original data** were encoded.

(b) If the given Hamming code was encoded using even parity, **it is possible to determine the original data bits** that were encoded. To derive these original data bits, we check if all of the parity bits are intact or if there are any corrupt bits:

Parity Bit 1 (Skip intevals of length 1)

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11
1		**1**		**1**		**0**		**0**		**1**		**1**		**1**

#1's = 6; even parity; no problem here

Parity Bit 2 (Skip intevals of length 2)

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11
	1	**1**			**1**	**0**			**0**	**1**			**0**	**1**

#1's = 5; odd parity; something's wrong with P2

Parity Bit 3 (Skip intevals of length 4)

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11
			0	**1**	**1**	**0**					**0**	**1**	**0**	**1**

#1's = 4; even parity; no problem here

Parity Bit 4 (Skip intevals of length 8)

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11
							1	**0**	**0**	**1**	**0**	**1**	**0**	**1**

#1's = 4; even parity; no problem here

Thus, bit position 2 has something wrong as per Hamming distance calculations. To figure out which bit was corrupted, we sum all the incorrect bit positions. Now the sum is 2 which means that it is not any data bit but the parity bit itself got flipped.

Corrected even-parity Hamming code

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11
1	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**

Thus, the original data bits are: 0b 1110 0010 101

(c) If the given Hamming code was encoded using odd parity, **it is still possible to determine the original data bits** that were encoded. To derive these original data bits, we check if all of the parity bits are intact or if there are any corrupt bits:

Parity Bit 1 (Skip intevals of length 1)

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11
1		1		1		0		0		1		1		1

#1's = 6; even parity; something's wrong with P1

Parity Bit 2 (Skip intevals of length 2)

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11
	1	1			1	0			0	1			0	1

#1's = 5; odd parity; no problem here

Parity Bit 3 (Skip intevals of length 4)

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11
			0	1	1	0					0	1	0	1

#1's = 5; even parity; something's wrong with P3

Parity Bit 4 (Skip intevals of length 8)

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11
							1	0	0	1	0	1	0	1

#1's = 4; even parity; something's wrong with P4

Thus, the bit positions 1, 4, 8 have something wrong as per Hamming distance calculations. To figure out which bit was corrupted, we add $1 + 4 + 8 = 13$. We have now detected that bit position 13 was corrupted.

Corrected odd-parity Hamming code

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	D10	D11
1	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**1**

Thus, The original data bits are: 0b 1110 0010 **0**01

3. (a) If memory page size = 16,384 = 2^{14}, the **page offset** in the virtual address space would occupy **14 bits (0, 1, ..., 13)**.

   Given the virtual address space is 32 bits long, the remaining bits apart from page offset: 32 - 14 = **18 bits (14, 15, ..., 31)** would be used for **virtual page number.**

   With 18 bits reserved for virtual page number, we can cover up to 2^{18} page numbers and thus there will be a total of 2^{18} = **262,144** entries in the page table.

   If one page table entry takes 5 bytes, the full page table would require a total of 262,144 x 5 = 1,310,720 bytes = **1.311 MB**.

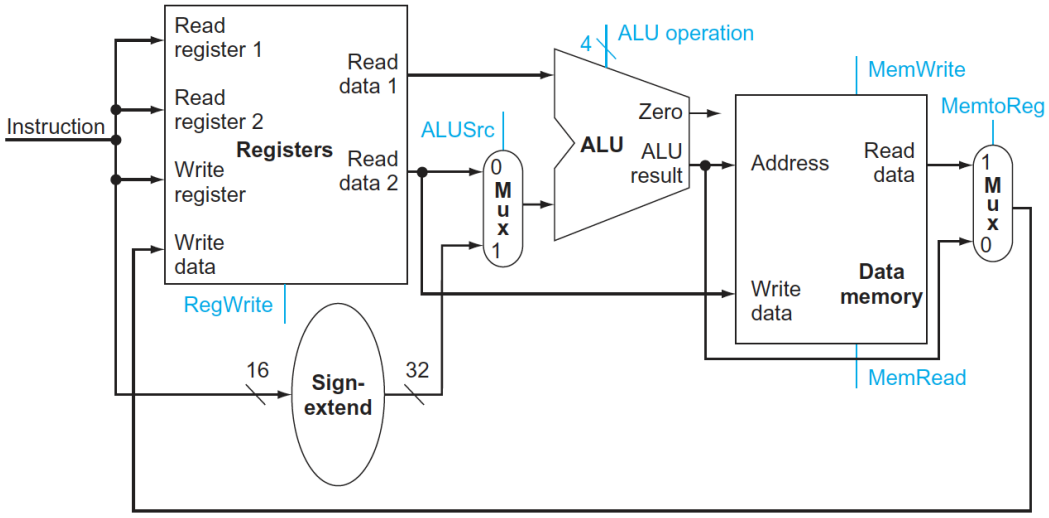
   (b) If memory page size = 16,384 = 2^{14}, the **page offset** in the virtual address space would occupy **14 bits (0, 1, ..., 13)**.

   Given the virtual address space is 48 bits long, the remaining bits apart from page offset: 48 - 14 = **34 bits (14, 15, ..., 47)** would be used for **virtual page number.**

   With 34 bits reserved for virtual page number, we can cover up to 2^{34} page numbers and thus there will be a total of 2^{34} = **17,179,869,184** entries in the page table.

   If one page table entry takes 5 bytes, the full page table would require a total of 17,179,869,184 x 5 = 85,899,345,920 bytes = **85.9 GB**.

4. Following the datapath with the given architecture:



(a) Instruction: `addi $15, $7, -36`

This instruction would be executed in the following flow:

i. The instruction is fetched and the program counter PC is incremented.
ii. Register $7 is read from register file and immediate field -36 is sent for sign extension.
iii. ALU operates on data read from register file and the 32-bit sign extended -36 to generate an addition.
iv. The result from ALU is written into the destination $15 inside register file.

The binary values of all control lines during this clock cycle are as follows:

i. **RegWrite: 1**
ii. **ALUSrc: 1**
iii. **ALU Control: 0010**
iv. **MemWrite: 0**
v. **MemtoReg: 0**
vi. **MemRead: X**

(b) Instruction: `lw $22, 76($10)`

This instruction would be executed in the following flow:

i. Instruction is fetched from the instruction memory and the PC is incremented.
ii. Register $10 value is read from register file.
iii. ALU computes the sum of the value read from the register file and the sign-extended offset (76).
iv. The sum from ALU is used as address for data memory.
v. Data from memory unit is written into register file at $22.

The binary values of all control lines during this clock cycle are as follows:

i. **RegWrite: 1**
ii. **ALUSrc: 1**
iii. **ALU Control: 0010**
iv. **MemWrite: 0**
v. **MemtoReg: 1**
vi. **MemRead: 1**

5. (a) The clock period for a pipelined implementation will be **330ps** as per the longest latency MEM stage.

   (b) The minimum clock period expected for a non-pipelined design is 250 + 270 + 190 + 330 + 225 = **1265ps** since each stage would run one after the other in sequence.

   (c) In order to improve the pipeline's performance, it makes most sense to try and reduce the longest latency stage somehow. If we try to split the MEM stage of 330ps into three stages, as per given caveat, we would end up **removing 330ps** and **adding 3 new stages of 40% of 330 = 132ps each**. Our new stages would then become:
   IF 250ps — ID 270ps — EX 190ps — MEM1 132ps — MEM2 132ps — MEM3 132ps — WB 225ps
   Thus, the **best stage to be split is the MEM stage** and the **newly pipelined CPU would now have a clock period of 270ps** due to ID being longest latency in the new pipeline. To fully execute an instruction however would now take 250 + 270 + 190 + 132 + 132 + 132 + 225 = **1331ps**.

   (d) Data memory is accessed by the Load & Store instructions. Thus, the utilization of the data memory is 18 + 17 = **35%** of the clock cycles when the data memory is actually being used.

   (e) The RegWrite signal is utilized during ALU and Load instructions in order to write a value into a register. Thus, the utilization of the write circuitry is 52 + 18 = **70%** of the clock cycles.