CS M151B: Homework 7

Jonathan Woong

804205763

Spring 2017

Discussion 1A

Monday 22nd May, 2017

Problem 1. In the system described in Ed2-Chap8 pp. 665-666, the memory system takes 200 ns to read the first four words, and each additional four words require 20 ns. The memory system is modified so that it takes 130 ns to read the first four words and 12 ns to read each additional four words, find the sustained bandwidth and the latency for a read of 256 words for transfers that use 4-word blocks and for transfers that use 16-word blocks. Also compute the effective number of bus transactions per second for each case.

For 4-word block transfers, each block takes

- 1 clock cycle to send address to memory
- $\frac{130\ ns}{5\frac{ns}{cycle}} = 26$ clock cycles to read memory
- 2 clock cycles to send data from memory
- 2 idle clock cycles between this transfer and next

This is a total of 31 cycles, and $\frac{256}{4} = 64$ transactions are needed, so the entire transfer takes $31 \times 64 = 1984$ clock cycles. Thus the latency is $1984\ cycles \times 5\frac{ns}{cycle} = 9920$ ns. The number of bus transactions per second is

$$\boxed{64 \text{ transactions} \times \frac{1\ second}{9920\ ns} = 6.4516\text{M transactions/second}}$$

The bus bandwidth is

$$\boxed{(256 \times 4)\text{bytes} \times \frac{1\ second}{9920\ ns} = 103.2258\text{MB/sec}}$$

For 16-word block transfers, the first block requires

- 1 clock cycle to send address to memory
- 130 ns or 26 cycles to read first four words in memory
- 2 clock cycles to send data of block, during which time the read of four words in next block is started
- 2 idle clock cycles between transfers, during which the read of next block is completed

Each of the three remaining 4-word blocks requires repeating only the last two steps. Thus, the total number of cycles for each 16-word block is $1+26+4\times(2+2) = 43$ cycles, and $\frac{256}{16} = 16$ transactions are needed, so the entire transfer takes $43 \times 16 = 688$ cycles. Thus the latency is $688\ cycles \times 5\frac{ns}{cycle} = 3440$ ns, which is roughly one-third of the latency for the case with 4-word blocks. The number of bus transactions per second with 16-word blocks is

$$\boxed{16 \text{ transactions} \times \frac{1\ second}{3440\ ns} = 4.6512\text{M transactions/second}}$$

The bus bandwidth with 16-word blocks is

$$\boxed{(256 \times 4)\text{bytes} \times \frac{1\ \mathit{second}}{3440\ \mathit{ns}} = 297.6744\text{MB/second}}$$

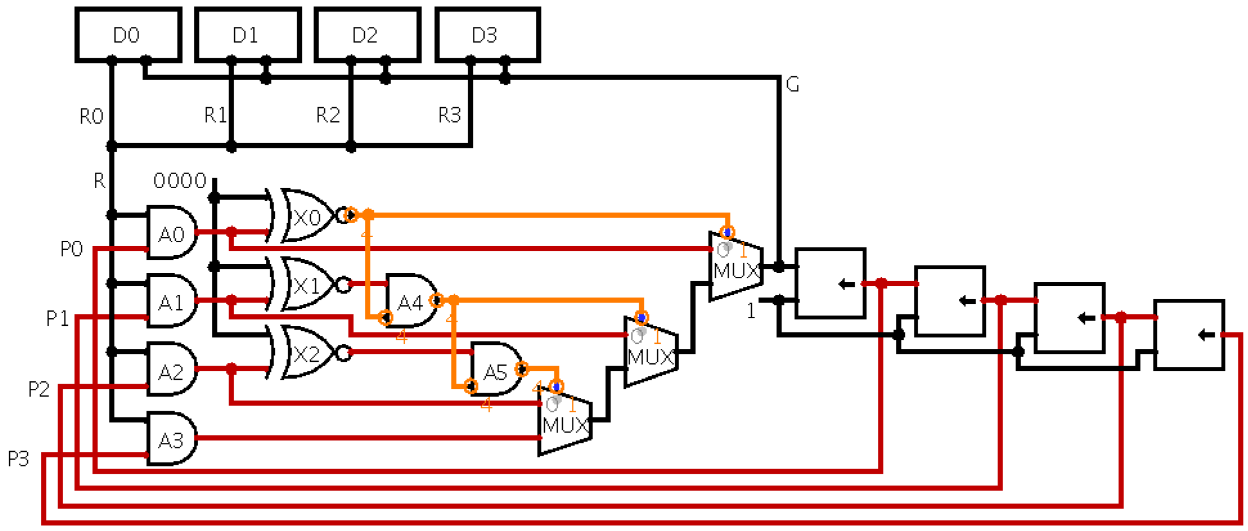
which is 2.88 times higher than the 4-word blocks.

Problem 2. Is there any situation where interrupt-driven I/O is prefereable to DMA-based I/O? Explain.

Yes, in situations where the CPU expects input from a device that is unpredictable (such as human typing on the keyboard), interrupt-driven I/O is preferable to DMA-based I/O. This is because the amount of time it takes to handle the interrupt signal of human typing requires immediate action from the CPU, and cannot be improved in response time if the CPU delegates the memory access to the DMA.

Problem 3. Design a "fair" bus arbiter. There are four devices connected to the bus. The arbiter will have four "*bus request*" lines as inputs. The outputs of the arbiter are four "*bus grant*" lines. When a device wants the bus it asserts its bus request line. A device gets the bus when its bus grant line is asserted by the arbiter. A device that gets the bus uses it for one cycle. The arbitration for any particular bus cycle occurs during the previous cycle. A device that is currently using the bus *can* contend for use of the bus during the next cycle. At most one device can use the bus during one cycle. Any number of devices (zero to four) can contend for the bus during every cycle. The arbiter enforces "fairness" among the devices by dynamically changing the relative priority of the devices. Denote the devices: D_0, D_1, D_2 and D_3. When the system is initialized, D_0 has top priority, then D_1 and D_2 with D_3 having the lowest priority. Note that even the lowest priority device may get the bus if none of the other devices request it. When some device $D_i (0 \leq i \leq 3)$ gets the bus, the top priority is changed to device D_j with $j = (i+1) mod 4$. The priority of devices is then $D_{(i+1)mod4}, D_{(i+2)mod4}, D_{(i+3)mod4}$, and $D_{(i+4)mod4}$.

A) Design the arbiter. Your design should be at the level of gates and flip-flops. Show your work (truth tables, etc.).



Combine each individual bus request signal (R_0, R_1, R_2, R_3) into a signal R, which is a 4-bit wide line with the left-most bit representing R_0 and the right-most bit representing R_3. Let R be one input to four AND gates (A_0, A_1, A_2, A_3), with the other input to this AND gate being the bit map representing the priority $P = (P_0, P_1, P_2, P_3)$ where P_0 is top priority and P_3 is lowest priority. Initially, P_0 is 1000 (D_0), P_1 is 0100 (D_1), P_2 is 0010 (D_2), and P_3 is 0001 (D_3). The output of A_0 will be 0000 if R_0 is 0, otherwise the output is P_0 (this applies to A_0, A_1, A_2, A_3). The XNOR gates are used to check whether an output of an AND gate is 0000, they take as input the output of an AND gate and the value 0000. For example, if the output of A_0 equals 0000, then the output of X_0 equals 1 (this is true for X_0, X_1, X_2). The output of these XNOR gates is the input to a MUX, which selects the highest priority non-zero bitmap to output as the grant signal G. G is then the input to a shifter which shifts G to the left by 1 (assume wrapping). The table below shows the execution of two cycles assuming $R = (0,0,1,1),(1,1,0,0)$:

R	P_0	P_1	P_2	P_3	A_0	A_1	A_2	A_3	X_0	X_1	X_2	A_4	A_5	G
0011	1000	0100	0010	0001	0000	0000	0010	0001	1	1	0	1	0	0010
1100	0100	1000	0001	0010	1000	0100	0000	0000	0	0	1	0	0	0100

B) Show a timing diagram that includes the clock, the bus request lines, the bus grant lines, and the bus data lines for five cycles under the following conditions:

- The system is intiailized before the first cycle.
- During the first cycle D_1 and D_3 request the bus.
- During the second cycle D_2 requests the bus.

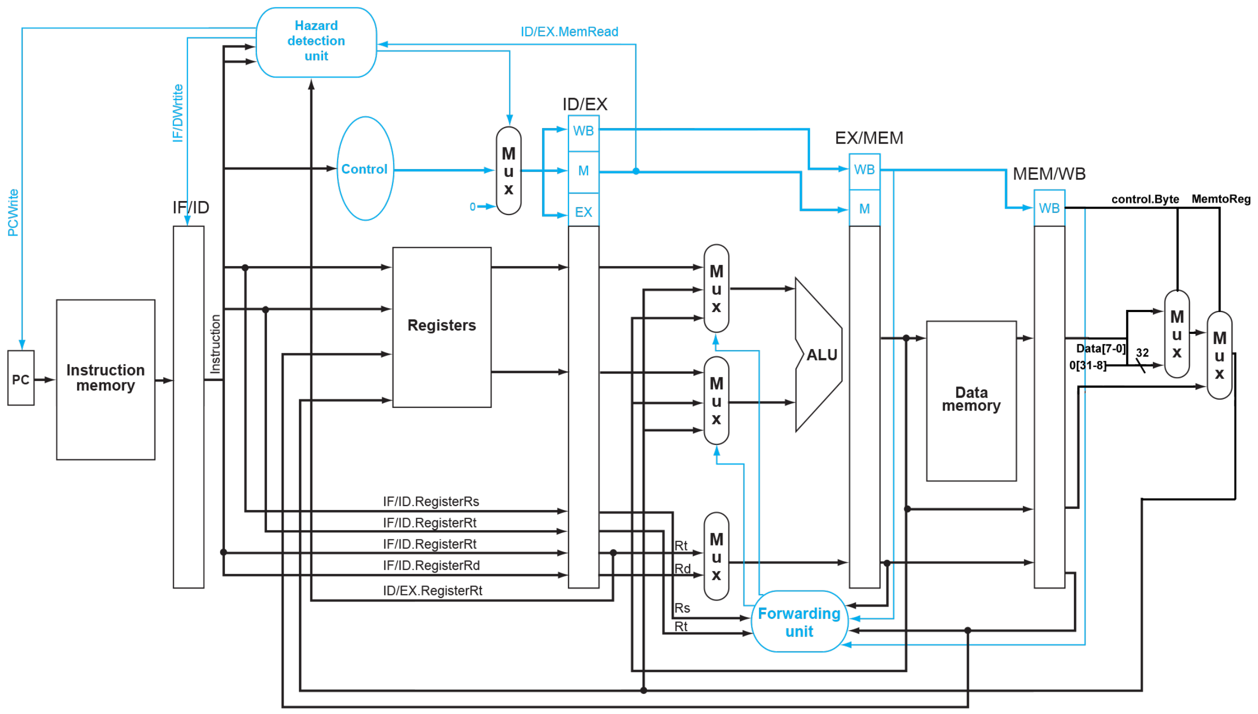
Clock	$R_3R_2R_1R_0$	$G_3G_2G_1G_0$	Data
1	1010	0000	
2	0100	0010	
3	0000	0000	D_1
4	0000	0100	
5	0000	0000	D_2

Problem 4. Recall that the MIPS ISA is big endian. Consider the pipelined MIPS implementation shown in Figure 4.60, page 316, with the enhancement for supporting `lw` and `sw` shown in Figure 4.57, page 312. In this implementation, the box labeled "Data memory" is a $2^{30} \times 32$ memory – a 32-bit word can be either read or written to the memory in every access. Hence, the address input to this memory consists of just 30 bits – the 30 most-significant bits of the effective address computed by the ALU. Your task is to add to this implementation support for the (sb) (store byte) instruction, documented on page A-68 in the book and page 2.22 in the class notes. You **cannot** modify the implementation of the data memory in **any way**. Be sure to consider any iterations between the new `sb` instruction and the already supported instructions that must continue to work.

A) Explain your modifications in 3-5 clear sentences.

- Add a new control signal `Byte` that is an input to a MUX.
- Add a MUX with the first input as the 32-bit word read from memory. The second input is the 8 LSB of the 32-bit word read from memory and with 24 bits of 0 appended to the MSB. When `Byte` is asserted, the second input becomes the output.
- The output of the `Byte` MUX is input to the MUX controlled by the `MemtoReg` signal.

B) On a copy of Figure 4.60, show the modifications.



C) List all the required new control signals with an explanation and identify with on the datapath figure.

Look at part A and B.

D) Changes are required to the main *Control* circuit. Show those changes using a table similar to the onw shown on slide 7.30 in the class notes.

Instruction	RegDst	AluOp	ALUSrc	Branch	MemRead	MemWrite	RegWrite	MemtoReg	Byte
R-format	1	10	0	0	0	0	1	0	0
`lw`	0	00	1	0	1	0	1	1	0
`sw`	X	00	1	0	0	1	0	X	0
beq	X	01	0	1	0	0	0	X	X
`sb`	X	00	1	0	0	1	0	X	1

Problem 5. Consider a $16M \times 1$ DRAM chip. The notation used below is: `read 100` is a read operation that reads one bit from address 100, `write 350` is a write operation that writes one bit from address 350. One of two possible sequences of operations are performed:

A) `read 50, read 500, read 5000`

B) `write 50, write 500, write 500`

What sequence of operations (A or B) can be completed faster? Explain your answer.

A $16M \times 1$ chip is implemented with a 4096×4096 array. This means `read 50` will cost 1 row access and 1 column access. Since `read 500` is in the same row as `read 50`, only 1 additional column access is required. `read 5000` costs 1 row access and 1 column access. The total number of accesses required by A is 5.

`write 50`, `write 500`, `write 500` are all in the same row, so the cost is 1 row access and 3 column accesses. The total number of accesses required by B is 4.

$\therefore$ B will complete faster.

Problem 6. You have $4M \times 8$ DRAM chips (as many as you want). You must use these chips to build the data memory for the single-cycle MIPS processor described in chapter 4 of the textbook. What is the minimum size memory you can build? Specify the size in bits. Explain your answer.

A $4M \times 8$ chip is implemented with a 2048×2048 array. Each row access will access 2K bits and throw away all but 8 bits. Since we can access 8 bits at a time, we will need a MUX that takes $\frac{2048}{8} = 256$ inputs. This means that the length of the address that acts as input to this MUX is 8 bits.

We can combine 4 of the $4M \times 8$ DRAM chips to handle accessing a 32-bit memory address. The first DRAM chip will take as input bits 0-7 of the address, the second DRAM chip will take as input bits 8-15, the third chip will take input bits 16-23, the last chip will take input bits 24-31. The 8-bit output of each DRAM chip are concatenated to represent the 32-bit word that is accessed.

The minimum size memory you can build is $4(2048 \times 2048) = 16M$ bits.