Department of Electrical and Computer Engineering

The University of Texas at Austin

EE 460N, Spring 2017
Problem Set 3 Solutions
Due: February 13, before class
Yale N. Patt, Instructor
Chirag Sakhuja, Sarbartha Banerjee, Jon Dahm, Arjun Teh, TAs

Instructions

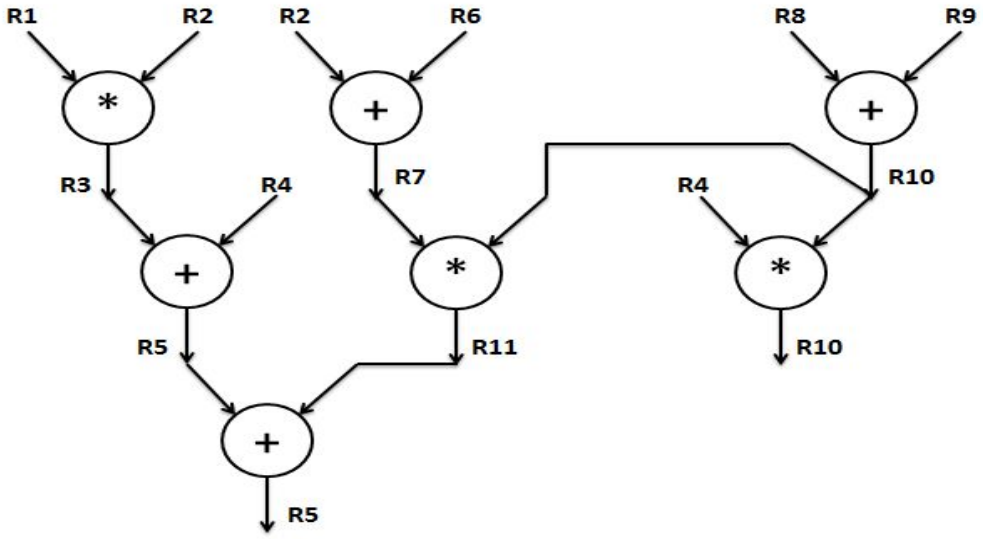
You are encouraged to work on the problem set in groups and turn in one problem set for the entire group. The problem sets are to be submitted on Canvas. Only one student should submit the problem set on behalf of the group. The only acceptable file format is PDF. Include the name of all students in the group in the file.

Note: You still need to bring a hard copy of your student information sheet to the class on Monday.

You will need to refer to the assembly language handouts and the LC-3b ISA on the course website.

Answers.

1.



2.

a.

1	ADD R7, R6, R7
2	ADD R3, R6, R7
3	MUL R0, R3, R6
4	MUL R2, R6, R6
5	ADD R2, R0, R2

b.

1	z	10	1	z	11	a
1	z	10	0	a	11	b
0	x	4	0	y	6	c

+

0	b	7	1	z	10	x
1	z	10	1	z	10	y
						z

X

c.

	V	tag	value
R0	0	x	4
R1	1	z	5
R2	0	c	6
R3	0	b	7
R4	1	z	8
R5	1	z	9
R6	1	z	10
R7	0	a	11

3. Truth table for the WE Logic:

MAR[0]	R.W	DATA.SIZE	WE1	WE0
0	RD	Byte	0	0
0	RD	Word	0	0
0	WR	Byte	0	1
0	WR	Word	1	1
1	RD	Byte	0	0
1	RD	Word	0	0
1	WR	Byte	1	0
1	WR	Word	0	0

```
RD = 0  WR = 1;  Byte = 0  Word = 1
WE0 = (MAR[0]') AND (R.W)
WE1 = R.W AND (MAR[0] XOR DATA.SIZE)
```

4.

Differences between exceptions and interrupts:

	Exceptions	**Interrupts**
Cause	Internal to the running process	External to the running process
When are they handled?	When detected (mostly)	When convenient (mostly)
Are they maskable?	Almost never	Almost always
Priority	Same as the process that caused the exception.	Depends (on the priority of the interrupting device)
Context	Process	Handling is done withing the system context.

Interrupts and exceptions are similar in how they are handled. To handle both, the machine has to be put into a consistent state and PC needs to be loaded with the address of the interrupt/exception handler.

Basic steps required to handle an exception/interrupt:

a. Exception/interrupt is detected.
b. The machine is put to a consistent state.

c. PC and PSR (Program Status Register) of the process are saved (on the stack).
d. PC is loaded with the address of the handler. (This address can be obtained by accessing the interrupt/exception vector table using the vector supplied by the event)
e. Interrupt/exception handler is executed.
f. Interrupt/exception handler returns back to the interrupted process. (PC and PSR of the interrupted process are restored)

5. a)
   - Byte on bus Addr[1:0]
   - Interleave bits Addr[4:2]
   - Chip address Addr[7:5]
   - Rank bits Addr[11:8]

b. 577 Cycles. The first 8 memory accesses, A[0][0] to A[0][7], must occur sequentially with no overlap since they are all acesses to the same bank. Thus, it would take 80 cycles for the 1st 8 memory accesses, with the 8th access starting in cycle 70. Since the 8th and 9th memory accesses, A[0][7] and A[1][0], respectively, are to different banks, the accesses can overlap, and the 9th access can start in cycle 71 (70 cycles for the 1st 7 accesses plus 1 additional cycle of the 8th access). Continuing with this logic, the access to A[2][0] could start in cycle 142 (71x2). Finally, the access to A[7][0] could start in cycle 497 (71x7). Now all that remains are 8 more memory accesses, all to the same bank (A[7][0] to A[7][7]). This takes another 80 cycles, bringing the total to 577 cycles (497 + 80).

c. If the memory were not interleaved, all 64 memory accesses must happen sequentially with no overlap, so it would take a total of 640 cycles (64*10). Therefore, we do gain some benefit from this interleaving scheme, but not that much.

Yes, a change can be made. The new bits are:
   - Byte on bus Addr[1:0]
   - Interleave bits Addr[4:2]
   - Chip address Addr[11:9]
   - Rank bits Addr[8:5]

73 Cycles. With the new interleaving scheme, consecutive memory accesses are to either to different banks of the same rank, or to different ranks all together. In both cases, the consecutive accesses can start immediately after each other. Therefore the latency of all memory accesses would be hidden except the first access. Total number of cycles = 10 + 63 = 73

d. Only one line of code needs to be changed:

sum = sum + A[i][j];
to
sum = sum + A[j][i];

Alternatively, you could keep that line the same, but swap the variable (i/j) of the inner and outer loops as shown below.

Original code:

```
for(i = 0; i < 8; ++i){
  for(j = 0; j < 8; ++j){
    sum = sum + A[i][j];
  }
 }
```

New code:

```
for(j = 0; j < 8; ++j){
  for(i = 0; i < 8; ++i){
    sum = sum + A[i][j];
  }
 }
```

87 Cycles. Now consecutive memory accesses are to different banks, so the accesses can overlap. The 1st access, A[0][0], would begin at cycle 0, the 2nd, A[1][0], at cycle 1, and so on. The 8th access, A[7][0], would start at cycle 7. However, the 9th access, A[0][1], cannot start at cycle 8. It would have to wait 2 more cycles for the 1st access to finish since it is on the same bank as the 1st access; therefore, it would start at cycle 10. Continuing this logic, the access to A[0][2] would start at cycle 20, and finally, the access to A[0][7] would start at cycle 70. Now, all that is left are 8 accesses, but they are all to different banks so they can start 1 cycle after each other. The access to A[1][7] would begin at cycle 71, A[2][7] at 72, and finally A[7][7], the last memory access, would begin at cycle 77 and, therefore, end at cycle 87

6.

In this problem, we assume that both of the rotators are right rotators. If the rotator for read is a right rotator and the rotator for write is a left rotator, PA[1:0] can be used as the control for both rotators.

Note: for WR (i.e. stores), the LD.MDR signal doesn't matter if the store takes only a single access, since we assume the MDR is already loaded with the data to be stored from the processor in the previous cycle. The only time LD.MDR matters is during the 1st access of a multi-access store; in this case, we need to make sure that the bytes that will be stored in the 2nd access aren't overwritten.

PA[1:0]	SIZE	RD/WR	1st/2nd	LD.MDR[3:0]	ROT[1:0]	WE[3:0]	sel
00	B	RD	X	XXX1	00	0000	0
00	B	WR	X	XXXX	00	0001	0
00	H	RD	X	XX11	00	0000	0
00	H	WR	X	XXXX	00	0011	0
00	W	RD	X	1111	00	0000	0
00	W	WR	X	XXXX	00	1111	0
01	B	RD	X	XXX1	01	0000	0
01	B	WR	X	XXXX	11	0010	0
01	H	RD	X	XX11	01	0000	0
01	H	WR	X	XXXX	11	0110	0
01	W	RD	1st	X111	01	0000	0
01	W	RD	2nd	1000	01	0000	1
01	W	WR	1st	0000	11	1110	0
01	W	WR	2nd	XXXX	11	0001	1
10	B	RD	X	XXX1	10	0000	0
10	B	WR	X	XXXX	10	0100	0
10	H	RD	X	XX11	10	0000	0
10	H	WR	X	XXXX	10	1100	0
10	W	RD	1st	XX11	10	0000	0
10	W	RD	2nd	1100	10	0000	1
10	W	WR	1st	0000	10	1100	0
10	W	WR	2nd	XXXX	10	0011	1
11	B	RD	X	XXX1	11	0000	0
11	B	WR	X	XXXX	01	1000	0
11	H	RD	1st	XXX1	11	0000	0
11	H	RD	2nd	XX10	11	0000	1
11	H	WR	1st	XX00	01	1000	0
11	H	WR	2nd	XXXX	01	0001	1
11	W	RD	1st	XXX1	11	0000	0
11	W	RD	2nd	1110	11	0000	1
11	W	WR	1st	0000	01	1000	0
11	W	WR	2nd	XXXX	01	0111	1

Legend:

B(yte)	00
H(alf word)	01
W(ord)	10
RD(read)	0
WR(write)	1
1st	0
2nd	1

7.

Interleave into 64 banks in order to hide the latency in sequential accesses (note, minimum needed is 37 banks but one would really prefer to use a power of 2, therefore 64).

8.

A. The probability of a single bit flipping is $p_f = 10^{-7}$. Therefore, the probability of a bit remaining correct is $p_c = (1 - 10^{-7})$. The probability that a transmitted nine bit message will have zero flipped bits is $p_c^9 = (1 - 10^{-7})^9$. Thus, the probability of at least one of the bits being flipped is $1 - p_c^9 = 1 - (1 - 10^{-7})^9 \approx 9 \times 10^{-7}$.

B. Parity check logic *can* detect an odd number of errors: 1, 3, 5, 7, and 9.

C. Parity check logic *cannot* detect an even number of errors: 2, 4, 6, 8.

D.

a. There are choose(9, 1) = 9 possible combinations that result in a single flipped bit. Thus, the probability of one bit being flipped is $p_1 = 9 \times p_f \times p_c^8 \approx 9 \times 10^{-7}$.
b. There are choose(9, 2) = 36 possible combinations that result in two flipped bits. Thus, the probability of two bits being flipped is $p_2 = 36 \times p_f^2 \times p_c^7 \approx 3.6 \times 10^{-13}$.
c. There are choose(9, 3) = 84 possible combinations that result in three flipped bits. Thus, the probability of three bit being flipped is $p_3 = 84 \times p_f^3 \times p_c^6 \approx 8.4 \times 10^{-20}$.

E. Ignoring the probability of three or more bit errors, the probability of a detected error is just the probability of a single bit error (calculated above). Thus, the rate of detected errors is $p_1 \times 10^9 \div 9 \approx 100$ errors per second.

F. Similarly, the probability of an undetected error is approximately the probability of a double error. Thus, the rate of undetected errors is $p_2 \times 10^9 \div 9 \approx 4 \times 10^{-5}$ corrupt messages per second (or twice as many undetected *bit* errors, since we assume each undetected corrupt message contains two flipped bits), which is equivalent to one undetected corrupt message about every 7 hours.

10.
Single Error. The corrected bit pattern:

- 111010010111

When there are two errors, there will definitely be a parity error. However, consider the case where P0 and P1 are the two errors. If we examine the parity errors with the intention of attempting to correct a single error as in the previous example, then we would erroneously think that the 3rd bit (D0) was in error. Clearly, this is not the case. It is also possible that after computing the parity errors, we determine, for example, that bit 15 is in error (all four parity functions evaluated incorrectly). However, the transmitted data only has 12 bits. Thus, if we see that an error points to bits 13 through 15, we know for sure that two errors occurred. However, as described earlier, it is definitely possible that two errors manifest themselves as a single error in one of the 12 bits transmitted.

More than two errors will manifest itself as a single error, two errors, or possibly even no error at all. This scheme has no way to distinguish more than 2 errors from 2 errors or less.