20th CSEC – Past Year Paper Solution (2018 – 2019 Sem 2)
CE/CZ1006 – Computer Organisation and Architecture

Solver: Tan Zarn Yao
Email: ztan059@e.ntu.edu.sg

1)

a) Note that SR is initially set to 0x001. Hence, operations such as MOV and OR will not affect the carry bit.

i) R1: 0x000, SR: 0x003
ii) R1: 0x002, SR: 0x001
iii) R1: 0xAE9, SR: 0x005
iv) R1: 0xD70, SR: 0x005
v) R1: 0x3C2, SR: 0x009

(b)

i)

	R0	R1	R2	R3	SP	SR
Initial Value	0x101	0xAE1	0x002	0x71F	0xFFB	0x001
MOVS R1, #0	0x101	0x000	0x002	0x71F	0xFFB	0x001
ROL R3	0x101	0x000	0x002	0xE3E	0xFFB	0x004
JNC Next	0x101	0x000	0x002	0xE3E	0xFFB	0x004
INC R0	0x102	0x000	0x002	0xE3E	0xFFB	0x004
CMP R0, #0x104	0x102	0x000	0x002	0xE3E	0xFFB	0x004
JLO Back	0x102	0x000	0x002	0xE3E	0xFFB	0x004
ROL R3	0x102	0x000	0x002	0xC7D	0xFFB	0x005
JNC Next	0x102	0x000	0x002	0xC7D	0xFFB	0x005
ADD R1, [R0]	0x102	0x0B0	0x002	0xC7D	0xFFB	0x000
INC R0	0x103	0x0B0	0x002	0xC7D	0xFFB	0x000
CMP R0, #0x104	0x103	0x0B0	0x002	0xC7D	0xFFB	0x004
JLO Back	0x103	0x0B0	0x002	0xC7D	0xFFB	0x004
ROL R3	0x103	0x0B0	0x002	0x8FB	0xFFB	0x005
JNC Next	0x103	0x0B0	0x002	0x8FB	0xFFB	0x005
ADD R1, [R0]	0x103	0x8B1	0x002	0x8FB	0xFFB	0x004
INC R0	0x104	0x8B1	0x002	0x8FB	0xFFB	0x004
CMP R0, #0x104	0x104	0x8B1	0x002	0x8FB	0xFFB	0x003
JLO Back	0x104	0x8B1	0x002	0x8FB	0xFFB	0x003
POP R2	0x104	0x8B1	0xFFF	0x8FB	0xFFC	0x003

ii) To solve this question, first we need to know the address of each instruction. Since the questions asked us to use the initial conditions from Figure Q1a and assume the instruction execution begins at the label Start, we know that the address of the instruction at label Start has the address 0x000 since PC value = 0x000 from Figure Q1a. From this, we can deduce that:

Address	Instruction
0x000	MOVS R1, #0
0x001	ROL R3
0x002	JNC Next
0x003	ADD R1, [R0]
0x004	INC R0
0x005	CMP R0, #0x104
0x007	JLO Back
0x008	POP R2
0x009	MOV R0, PC
0x00A	MOV PC, [0x104]
0x00C	NEG R2
0x00D	JMP Loop

1. MOV R0, PC will move the start address of the next instruction into R0. Hence, R0 becomes 0x00A. SR will change to 0x001 since N and Z flags will be set to zero while C flag is unaffected.
2. MOV PC, [0x104] will move the value in the address 0x104 into PC. Hence, PC value changed to 0x100. SR value remains to be 0x001 since 0x100 is not negative and non-zero.
3. Instruction at the address 0x100 will be executed, i.e. 0x40C which means ADD R0, #n, and the immediate value is stored in the next address (0x101). Hence, the instruction is read as ADD R0, #0x002. R0 becomes 0x00C. SR becomes 0x000.
4. PC value is now 0x102 and the instruction 0x0B0 is executed. The instruction is read as MOV PC, R0. The value in R0 is moved into PC. PC changes to 0x00C. SR is still 0x000 since 0x00C is not negative and non-zero.
5. Instruction at the address 0x00C is executed, ie. NEG R2. The value in R2 becomes 0x001. SR remains to be 0x000 since it is non-negative, non-zero and there is no carry.
6. JMP Loop is executed. PC value jumps to the address at the label Loop, ie 0x00C.
7. NEG R2 is executed again. R2 becomes 0xFFF. SR becomes 0x004 since 0xFFF is negative.
8. JMP Loop is executed again.
9. This will form an infinite loop where the value of R2 changes between 0x001 and 0xFFF and the value of SR changes between 0x000 and 0x004.

2)

a)
```
if (R2 < 0) {
      R2 = -R2;
}
R0 = R1;
```

b)

I1	PSH #0x202
I2	PSH [0x200]
I3	PSH [0x201]
I4	PSHM 15
I5	MOV R1, [SP+6]
I6	MOV R2, [SP+5]
I7	MOV R3, [SP+7]
I8	MOV [R3+0], R0
I9	POPM 15
I10	RET

c) $Nz = Nx * 2^{|Ny|}$

- With each iteration of Ny, Nx -> 2Nx -> 4Nx -> 8Nx -> 16Nx

$Nz = 18 * 2^5 = 576 = 0x240$

d)

	MOV AR, R2
Loop	ADD R0, R1
	ADD R1, R1
	JDAR Loop

3)

a)

i) Storage Memory should be non-volatile since we will want to retain the data in a Notebook PC even when electric power is removed.

ii) NOR Flash since the NOR Flash supports Execute in Place which means that the bootloader code in the Internal Flash can be executed directly without having to transfer to RAM first.

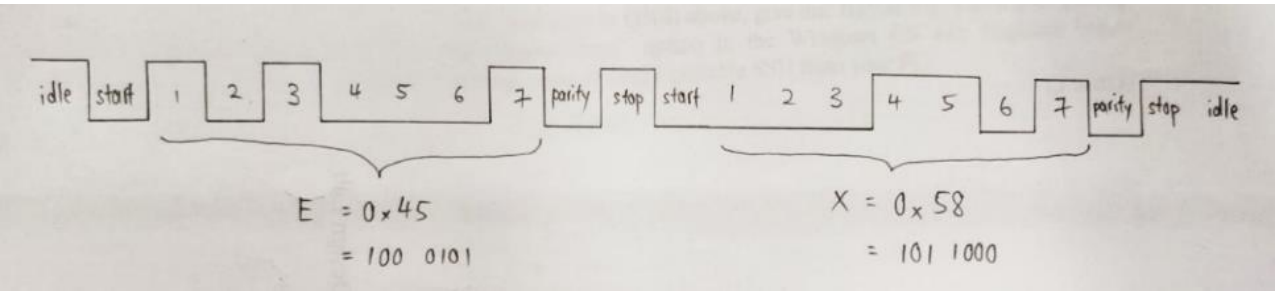
iii) No, since SRAM is volatile in nature and the bootloader code will be lost when the PC is shut down (i.e. no more electric power). In this case, we will not be able to transfer the Operating System code from the Storage Memory to the System Memory the next time the PC is booted.

iv) HDD is not suitable because this product is designed to be able to sustain drop from 1.2 meters above ground. HDD which consists of moving mechanical parts is more prone to crashing and will not be able to sustain a drop from such height.

b) 1. Less issue with signal skew and less probability of being influenced by crosstalk.
2. Less bulky so easier to route PCB trace and cabling.
3. Modern serial standards can achieve relatively high data transfer speed even though data is transferred one bit at a time.

USB replacing Parallel Port Interface, SATA replacing IDE

c) 1 Start bit, 7 Data bits (LSB first), 1 Stop bit, 1 Odd Parity bit



Assumption: The RS232 interface uses the UART protocol.

d) The following techniques are used to extend the life of SSD:
1. wear levelling, which is to distribute data and erase/write cycles evenly over the entire disk
2. use of external RAM as a buffer to minimise the number of writes to Flash in SSD.

e) If an external RAM is used as a buffer in SSD and the "Eject Drive" option is not executed before attempting to unplug the SSD, the RAM might not have enough time to store its data to Flash in SSD. This may lead to data loss since RAM is volatile in nature.

4)

a) TLB is a page table cache located in the internal fast memory close to CPU which is used to store a list of most recent page lookup values. It indicates which pages have been recently accessed.

The design is bad because the page size is too big. The physical memory can only hold up to 2^5 pages (physical memory size/page size = $2^{16}/2^{11}$), which means that the Page Table can only have 2^5 entries, which is equal to the size of TLB. In this case, the TLB is not used as the number of entries in the TLB is the same as the Page Table, and every time a TLB miss occurs will lead to a page fault. Access to secondary memory is required hence leads to longer access time and decrease in system performance. Given the virtual memory size of 1Mbyte, we can have 2^9 virtual pages (virtual memory size/virtual page size = $2^{20}/2^{11}$), and a page table with only 2^5 entries implies that there is a 93.75% chance that a page fault will occur:

[Number of virtual pages – page table entries]/number of virtual pages x 100% = [29 -25]/29 x 100%

This will largely degrade the CPU performance.

b)

i) FIFO policy replaces the block that has been in the cache the longest, regardless of when it was last used. LRU policy replaces the block that has been unused for the longest period of time.

ii) Since the cache block is of the size of 16-Byte, it means that we need 4 LSBs to be the offset, and the 4 MSBs to be the tag. 0xA1, 0x34, 0x22 and 0x60 each is placed in a different cache block since their tag value is different. 0x28 will not replace any cache block as its tag value is the same as 0x22 and the data will be accessed with an offset value of 8 at the corresponding cache block.

However, when 0x71 arrives, it has a different tag value from all the cache blocks in the cache memory and the cache memory is now full. Hence, it will need to replace a cache block.
If FIFO is used, 0xA1 will be replaced since it has been in the cache the longest.
If LRU is used, 0xA1 will also be replaced since it has been unused for the longest period.

When 0x42 arrives, it also needs an empty block in the cache since its tag value is different.
In this case: If FIFO is used, 0x34 will be replaced since it has been in the cache the longest.
If LRU is used, 0x34 will also be replaced since it has been unused for the longest period.

Hence, the user will not be able to differentiate whether the cache is using a FIFO or LRU cache replacement policy. A simple modification is to touch 0xA1 again just before 0x71, i.e.:
0xA1, 0x34, 0x22, 0x60, 0x28, 0xA1, 0x71.

If FIFO is used, 0xA1 will be replaced since it has been in the cache the longest.
If LRU is used, 0x34 will be replaced since it has been unused for the longest period.

c) The 32-bit floating number can have a much larger range because the radix point is able to float freely between the bits while the radix point is fixed in a position in the 32-bit fixed point number hence some of the bits may not be fully utilized. Precisions are traded off to achieve the greater range in the floating-point number representation.

d)

i) To resolve the data conflict between instructions I2 and I4. Imagine if there is no NOP instruction:

I2	F	D	E	**S**	
I4		F	D	**E**	S

The value 200 has not been stored in R0 when I4 is trying to retrieve the value from R0.

ii)

Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
I1	F	D	E	S											
I2		F	D	E	S										
I3			F	D	E	S									
I4				F	D	E	S								
I5					F	D	E	S							
I6						F	D	E	S						
Assume first branch is TRUE, hence I4 is loaded as the next instruction.															
I4							F	D	E	S					
I5								F	D	E	S				
I6									F	D	E	S			
Branch result of iteration #1 is TRUE, hence assume iteration #2 is also TRUE. I4 is fetched next.															
I4										F	D	E	S		
I5											F	D	E	S	
I6												F	D	E	S

Continue until iteration #10

I4	F	D	E	S						
I5		F	D	E	S					
I6			F	D	E	S				
Branch result of iteration #10 is TRUE, hence assume iteration #11 is also TRUE. I4 is fetched next.										
I4				F	D	E	S			
I5					F	D	E	S		
Branch result calculated to be FALSE. I4 and I5 are flushed. I7 is fetched next.										
I7						F	D	E	S	
I8							F	D	E	S

From the table, we know that each iteration of branch takes 3 cycles and the final iteration of branch resulted in 2 wasted cycles. Hence, the total number of cycles required to execute the entire program is:

I1-I3: 3 cycles
I4-I6: 3 cycles x 10 + 2 cycles
I7-I8: 5 cycles
Total: 40 cycles

==End of Answers ==