CE/CZ1006 University Library

CSEC 17th - Past Year Paper Solution *2016-2017 Sem1*
CE/CZ1006 Computer Organization and Architecture

Solver: Qiu Haoze

Email Address: qiuh0008@e.ntu.edu.sg

1.

a)

(i) `MOV R2, #0xFF1`

Answer: R2 **0xFF1** SR **0x004**

(The N flag of SR will be set since `0xFF1` is a negative number)

(ii) `MOV R2, [R3+3]`

Answer: R2 **0x000** SR **0x002**

(The content of memory location `0x204`(`0x201+3`), which is `0x000`, will be moved into R2, and the Z flag of SR will be subsequently set)

(iii) `EOR R2, [0xFF2]`

Answer: R2 **0x932** SR **0x004**

(
`0x823` = 1000 0010 0011
EOR `0x111` = 0001 0001 0001

1001 0011 0010 = `0x932`

The N flag of SR will be set since `0x932` is a negative number)

(iv) `SUB R2, R1`

Answer: R2 **0xE25** SR **0x004**

(Here we do R2 + (-R1) instead of R2 – R1, and because of the negative result, the N flag of SR will be set)

(v) `POPM 5`

Answer: R2 **0x000** SR **0x000**

(In the `POPM` instruction, bits 0-3(in this case it's 5 = 0101) corresponds to R0-R3, the stack pops to the corresponding register (in this case R0 and R2) of the set bit(s) in the order of R0 first and R3 last. Therefore, `0x70F` will be popped into R0 first, then `0x000` into R2.

`POPM` instruction does not affect SR, so SR remains 0x000 after the exection)

(b)

(i)

```
RETURN    POP R2          ;step1
          BSSR 1          ;step2
LOOP      RLC R1          ;step3,6
          ADD R0,#0xFFF   ;step4,7
          JNE LOOP        ;step5,8
MIDWAY    ADD R3,R2       ;Answer
```

Step/Reg	R0	R1	R2	R3	SR	SP
0	0x002	0x9FE	0x823	0x201	0x000	0xFF3
1	0x002	0x9FE	**0x70F**	0x201	0x000	**0xFF4**
2	0x002	0x9FE	0x70F	0x201	**0x001**	0xFF4
3	0x002	**0x3FD**	0x70F	0x201	0x001	0xFF4
4	**0x001**	0x3FD	0x70F	0x201	0x001	0xFF4
5	0x001	0x3FD	0x70F	0x201	0x001	0xFF4
6	0x001	**0x7FB**	0x70F	0x201	**0x000**	0xFF4
7	**0x000**	0x7FB	0x70F	0x201	**0x003**	0xFF4
8	0x000	0x7FB	0x70F	0x201	0x003	0xFF4
Answer	0x000	0x7FB	0x70F	**0x910**	**0x00C**	0xFF4

(ii)

```
MIDWAY    ADD R3,R2     ;Last Answer
NEXT      MOV PC,#0x201
          PSH R1
          SUB R3,R3
FINISH    RET
```

	Content
0x200	0x032
0x201	0x43C
0x202	0x001
0x203	0xBFD
0x204	0x000
	:

Answer: After the execution of `MOV PC,#0x201`, the program will proceed to memory address `0x201`. The content of it is `0x43C`, which means `MOV R3, #0x001`(the immediate value is from `0x202`) in machine code, then the content of R3 will be `0x201`, setting SR to `0x000`

Next, the program proceed to `0x203`, in which we have a `JMP -3`, then the program will jump back to `0x201`. Then everything happened just now will happen all over again and again in a infinite loop. All in all, R3 will be changed to and remain `0x001`, SR will be set to `0x000`, and PC will run in an infinite loop of 0x201 > 0x203 > 0x204 >0x201.

2.

a)

```
Main    :
        MOV     [0x101],#0
        PSH     [0x100]         ;Push value in memory variable N to stask       (I1)
        PSH     #0x101          ;Push the address memory variable Ans to stask  (I2)
        CALL    SumN
        ADD     SP,#2           ;Remove parameters on stack                     (I3)
          :
          :
SumN    PSHM    0x003
        MOV     R0,[SP+4]       ;Retrieve the value of N from the stack into R0 (I4)
        MOV     R1,[SP+3]       ;Retrieve the address of Ans from stack into R1 (I5)
          :
        CALL    SumN
          :
        RET                     ;Return to calling program                      (I6)
```

b)

```
SumN            PSHM    0x003           ;Save away used registers
                MOV     R0,[SP+4]       ;Retrieve the value of N from the stack into R0
                MOV     R1,[SP+3]       ;Retrieve the address of Ans from stack into R1
                ADD     [R1],R0         ;Add the current value of N to Ans
                DEC     R0              ;Decrement N by 1
                JEQ     FINISH          ;Jump to FINISH if N is 0
                PSH     R0              ;Push N to stack
                PSH     R1              ;Push the address of Ans to stack
                CALL    SumN            ;Start a new recursive call
                POPM    0x003           ;This step is just to clear stack
FINISH          POPM    0x003           ;Retrieve R0 and R1 of the Calling program
                RET                     ;Return to calling program
```

Tips:

1) In line 7 and 8, two `PSH`s are used instead of `PSHM`, that is because `PSHM` pushes in a different order(R1 first) and this will cause line 2 and 3 to retrieve N to R1 and the address of Ans to R0.

2)Line 10 and 11 are intended to clear previously saved stack. The FINISH label is put in line 11 because in the last recursive call, only 2 registers are pushed to stack (by the instruction in line 1).

c)

```
LOOP   CMP    [Var1],[Var2]
       JGE    SKIP
       SUB    [Var2],[Var1]
       JMP    LOOP

SKIP     :
```

3.

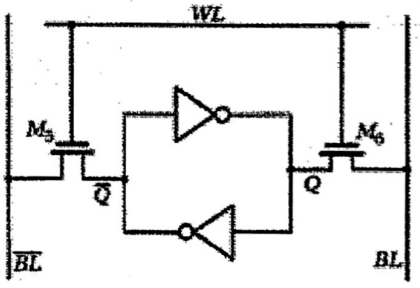
a)

Requirement	Memory Type
Memory to implement Translation Lookaside Buffer	SRAM
64GByte USB Thumb Drive	NAND Flash
Storage memory for a Network Access Server used to handle local cloud and backup functions at home	Magnetic HDD

Explanation:

i) TLB requires super fast speed, which is one of the main feature of SRAM. In spite of the high cost per bit of SRAM, TLB does not need very much space to implement so we can afford to use it.
ii) NAND Flash, with small physical size and relatively low cost per bit, is suitable for a mini drive with large storage space.
iii) Magnetic HDD has very low cost per bit and almost infinite erase cycle, which is perfect for backup storage.

b)



When writing, WL will be set to HIGH so that current from BL and $\overline{BL}$ can go through them. BL will be the the bit you want to write(1 or 0), and $\overline{BL}$ will be BL inversed. Therefore, the logic level of Q and $\overline{Q}$ will be the same respectively with BL and $\overline{BL}$. WL will then be set to LOW. With the two inverter in the centre, the logic level at Q and $\overline{Q}$ will be maintained as long as the power is on.

When reading, WL will be set to HIGH and previously written logic level will then be on BL and $\overline{BL}$.

The pass transistors are intended to control write & read operations through word line.

c)

Electrical signal level should be the primary consideration when connecting two electrical device togeter, output voltage level of output device do not exceed the maximum input voltage level of the input device.

Second, they must use the same communication protocol in order to communicate with each other. That is to say, one packet should mean the same thing for both devices.

d)

i)

the maximum number of packet A1 can transfer = 1s/(20μs*25 + 500μs)
= 1000 packets/sec
the maximum baud rate of A1 = bits per packet * 1000packets/sec
= 10*1000 = 10000

It becomes trickier when it comes to A2. We can compute the maximum baud rate supported by A2 by comparing the minimum delay between requests and time needed to transfer one packet.

Time needed to transfer one packet
= Switching from CPU to DMA + System Bus speed * 2(Because it's Fetch and Deposit DMA, data will be fetched to DMA first, then to its destination) + Switching from DMA back to CPU
= 100 μs + 50 μs * 2 + 100 μs = 300 μs

which is shorter than the minimum delay between requests, so the baud rate of A2 is restricted by the minimum delay. Therefore:

the maximum number of packet A2 can transfer = 1s/500μs
= 2000 packets/sec
the maximum baud rate of A2 = bits per packet * 2000packets/sec
= 10*2000 = 20000

From this we can conclude that the maximum baud rate allowed using cycle-stealing will be 9600(The biggest allowed that is smaller than 10000).

ii)

The answer will be 19200. That is because burst mode allows A1 to dedicate in data transmission fully, shorten time needed to transmit 1 packet to 500 μs, then the max baud rate of A1 will be 20000. Burst mode makes A2 to have a larger max baud rate than 20000, but the speed of the whole system will be constrained by A1. Therefore, the max baud rate will be 19200.

iii)

This may potentially cause data with integrity problem to be used as good data, because there is no way to detect corrupted packets.

4.

a)

Wear levelling is a technique for prolonging the life span of flash memory, which is used in solid-state drives (SSDs) and USB flash drives. It is needed because flash memory has limited erase cycle, which means that without a memory management scheme, flash memory may not be long-lasting.

b)

The effective access time of virtual memory = 10 ns * 80% + (10ns + 50 ns) * 20%
= 20 ns

The effective access time of cache = 5 ns * 90% + (5 ns + 20 ns) * 10%
= 7 ns

c)

The question says the result needs to be rounded to a fixed-point integer number. However, I was really confused by the term 'fixed-point integer number', and failed to find such a data type on Google. Therefore, I will assume it to mean just fixed-point number with scaling factor of 1(just like integer).

1. Convert all the fixed-point numbers' scaling factor to 2(1 digit after the decimal point).
   (This is to align the numbers to the same scaling factor. Addition and subtraction operations require the numbers to be aligned)

2. If the last truncated digit is 1, add 1 to the last digit.
   (This step is rounding, in order to achieve better accuracy)

3. Carry out the operation.

4. Convert all the results' scaling factor to 1(integer), if the truncated digit is 1, add 1 to the last digit.

d)

i)

1. Resourse conflicts: two instructions attempt to access the same resource in the same cycle.

2. Data conflict: the destination register of an instruction is also the source operand of its next instruction. In a pipelined processor, the result of the first instruction is not updated yet by the time it is fetched by the second instruction.

3. Branch delay: by the time the branch target is calculated, unnecessary instructions could have been fetched and decoded, then the unnecessary instructions would have to be flushed away, which cause significant performance loss.

ii)

1. R1 is the destination register of `I3` and referenced by the source operand of `I4`, causing a data conflict. This can be resolved by inserting a `NOP` instruction after `I3` to stall the pipeline.

2. By the time the result of I4 is stored to [R0], the content [R0] will be fetched by I7, causing a Resource conflict. This can be resolved by either delaying I7 by one clock cycle or adding new recourses.

3. The execution of I6 can cause the fetched and decoded I7 and I8 to be flushed away, causing a branch delay. This can be resolved by delayed branching, that is to put I6 before I4, since AR will not be effected by the execution of I4 and I5. In this way, by the time the branch is executed, there will be no extra instruction loaded, therefore no branch delay will occur

All assembly code in this solution is verified by VIPAS.

Please pardon me if there is any error, it has been a long holiday and my memory went a bit rusty!

For reporting of errors and errata, please visit pypdiscuss.appspot.com

Thank you and all the best for your exams! ☺