Question 1

1)

a)

R1	SR
0xFF9	0x004
0x000	0x002
0x121	0x000
0x873	0x00C
0x89B	0x005

b)
```
START   PSHM 3              ; Avoid using R2,R3, push fewer registers
        MOVS R0, #0          ; Reduce instruction cycle
        MOV AR, #23          ; Use AR register for looping
        MOV R1, [0x100]      ; The value at address 0x100 is used multiple times
LOOP    ADD R0, R1
        JDAR LOOP
```

The given code adds the value stored at address 0x100, which is 2, and the value of R1, 23 times (once at the start of the loop, 21 more times in the loop, once more at FINISH). In this optimized code, the AR register is involved. JDAR is the instruction that uses the self-reducer AR as the condition for jump.

JDAR is equivalent to:

```
AR -= 1
IF AR == 0:
    BREAK
ELSE:
    GO TO LOOP LABEL
```

c) R0 is added with 2 22 times in the loop, and once more at FINISH. Thus, R0 = 23 x 2 = 46

d) The position dependent points would be:

1. The literal address 0x100
2. Restore the content of register is missing

Therefore, we assume the data section is started at the memory address 0x100. We change this address to a parameter that is passed into the subroutine. Moreover, we add the recovery instruction so that the content of R0-R3 is restored after the execution of the subroutine.

```
        PSH #0X100
START   PSHM 3
        MOVS R0, #0
        MOV AR, #22
        MOV R1, [SP+3]
LOOP    ADD R0, R1
        JDAR LOOP
        POPM3
```

Question 2

2)

a)

i) The push R2 means we are using R2 as a temporary register in this subroutine. Therefore, we want to save its content, in order to recover the content after the finish of this subroutine.
We subtract SP by 3 to create a local frame in stack. In this case, we create three local variables in stack.

ii)

SP->	Base -6	Local_3
+1	Base -5	Local_2
+2	Base -4	Local_1
+3	Base -3	R2
+4	Base -2	Return address
+5	Base -1	Address of Var_X
+6	Base	Var_N

From the memory map, we can know the address of parameter Var_N: SP + 6, the address of Var_X: SP + 5.
b1: MOV R2, [SP + 6]
b2: MOV R2, [SP + 5]

b) c1: ADD SP, #3
POP R2
RET

d1: ADD SP, #2

c) Assume the local frame variables have the name as shown in the memory map above. By observation, X, Y, Z are two words variables. In this case, we can assume R2 is a pointer that points to a two words variable.
Let's find the algorithm of the assembly code:

```
SubA  PUSH R2              Save content of R2
      SUB SP, #3           Create local frame
      MOV R2, [SP+6]       R2 = N = 3
      MOV [SP], R2         Local_3 = N = 3
      MOV R2, [SP+5]       R2 = &X
      MOV [SP+2] , #0      Local_2 = 0
      MOV [SP+1], #0       Local_1 = 0
Loop  CMP [R2], #0         while(1){
                                 if(*R2<0)
      JPL Skip                   {
      INV [R2 +0]                    find the 2's complement of R2
      INV [R2+1]                 }
      ADD [R2+1], #1
Skip  ADD [SP+2], [R2+1]         [Local2: Local1] += *R2
      ADDC [SP+1], [R2+0]
      SUB [SP], #1               N--;
      JEQ Done                   if (N==0) break;
      ADD R2, #2                 R2++;
      JMP Loop             }
Done  MOV [R2+0], [SP+1]         Place the summation into Z
      MOV [R2+1], [SP+2]
```

We are able to know that the code does a reduce operation with the function:

```
if num< 0:
    num =- num;
sum += num
```

One more thing: mind the big endian and small endian representation. X, Y, Z, [Local2:Local1] are big endian,

Var_N = 0x003, Var_X = 0x111FFF, Var_Y = 0x600002,
Var_Z = 0x100222 + 0x600002 + 0x111FFF = 0x812223
The memory map:

0x100	0x003
0x101	0x111
0x102	0xFFF
0x103	0x600
0x104	0x002
0x105	0x812
0x106	0x223

d) One of the parameters is passed by reference: the base address of a 2 words array, and another parameter is passed by value: the size of the array. Passed by reference allows the subroutine to take more parameters, and the data size of each parameter can be larger. Thus the subroutine can perform operations over large amount of data.

Var_X, Var_Y and Var_Z are interpreted as the elements of a 24bits array.

3)

a) 1. NAND Flash
Given the features of the memory: robust, large capacity and non-volatile. SRAM, DRAM do not satisfy non-volatile properly. EEPROM (roughly a few megabytes) and NOR flash (roughly 64MB to 2GB) do not satisfy large capacity as they are around few megabytes. NAND is more robust to movement compared to HDD.

2. EEPROM
SRAM, DRAM should be out as they are volatile memory. A common mistake for this question is that student may choose flash memory. However, they ignore an important signal: *Each set data is < 64 bytes and can be erased without affecting other sets of user configuration*. This means the erase block size is < 64B. Flash memory has an erase block size ~4KB, which is way larger than required size. EEPROM has an erase block size ~16B, which is suitable.

3.Magnetic HDD
Storage memory is secondary memory, requiring cheap and non-volatile storage device. Data center frequently involves data exchange, therefore, NOR flash and NAND flash are not suitable with certain W/R lifecycle. HDD is cheap and has nearly infinite W/R times, which is suitable for storage memory.

b) **Serial**
Serial is suitable for long distance data transfer, as parallel needs more data lines which is costly if the distance is long. Moreover, parallel transfer will suffer from serious signal skew and crosstalk, reducing the integrity of the data.

Synchronous
Transfer can be done at a higher clock rate as a clock signal is used to strobe the data signal.

Differential signalling
Using differential signal will increase the resistance of the signal, preventing from noisy environment.

c)

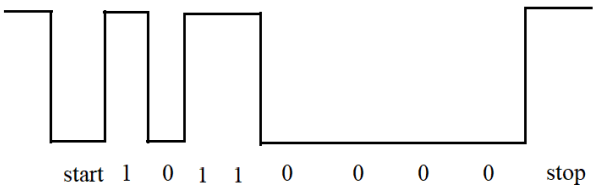


Fig1. COMP-B transmit

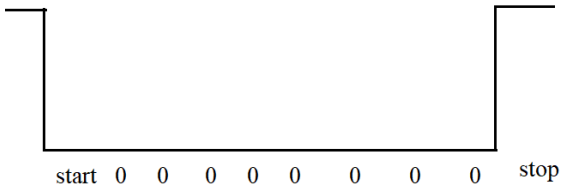


Fig2. COMP-A receives

From the two figures above, we can see that COMP-A samples the start bit of COMP-B 9 times. Hence, the baud rate of COMP-A is at least 9 times faster than the baud rate of COMP-B. Thus, possible baud rates of COMP-B are 9600, 4800, 2400.

d) 1.No moving parts, more robust to movement
2.Lighter and copy less space
3.Higher data transfer rate

Question 4

4)

a) **Write through**
Updates cache and main memory simultaneously on every write
Write back
Updates memory only when the block is selected for replacement

Write back policy should be chosen as write through will occupy the already limited system bus bandwidth more than write back.

b) **Virtual to physical:**

As page size is 1KB, the offset for each page will be 10 bits. The page number for a virtual address 0x00911 would be 0000 0000 10 -> 2

Page 2 is mapped to frame 8. Therefore, the physical address is 0000 0010 00 = **0x02111**

MM to Cache:

Use the main memory address 0x2111 we can map the MM to cache.

#cache block = 16, therefore, #block bits = 4 bits

Block size = 16B, therefore, #offset bits = 4bits

64KB physical memory, therefore #address bits = 16

Thus, #tag bits = 16 – 4 – 4 = 8

The cache block would be block 1.

Tag address would be 0010 0001 -> **0x21**

c) **Carry:** Set when the register cannot properly represent the result as an **unsigned** value (no sign bit required).
Overflow: Set when the register cannot properly represent the result as a **signed** value (you overflowed into the sign bit)

Function:
Carry:

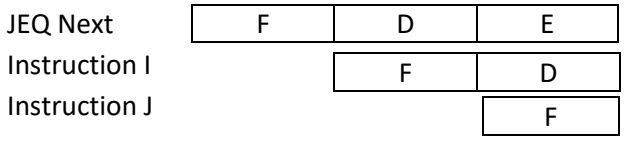
1. Combined with the second highest carry bit to detect overflow.
2. Use as the additional bit in order to perform addition of big number that is larger than the size of the register.

Overflow:

1. Use to detect out of range, so that wrong result can be notified.

d)

i)



The branch target is only known after the Execute stage but by this time, instruction I and J have already been fetched. Thus, there is a branch conflict.

ii) Data Dependency
I2, I3: R1 will not be ready at the execution stage of I3.
I8, I9: R2 will not be ready at the execution stage of I9.

Branch

I7, I8, I9: Target of the branch will not be ready before the fetch and decode of its following instructions. Those unwanted instructions will be discarded.

==End of Answers==

Solver: Li Yuanming