1/ op1: max: LODL 1 ; op2

op2: SUBL 2 ; op2-op1

opres: 0 JNEG op1big:

.LOC 50 LODL 1

main: LODD op1: RETN ; op2

PUSH op1big: LODL 2

LODD op2: RETN ; op1

PUSH

CALL max:

INSP 2

STOD opres:

HALT

2/ main: LODD index

DESP 2 ; sp:=sp-2;

LOCO 0 ; ac:=0

STOL 0 ; m[sp+0]:=ac (sum:=0)

STOL 1 ; m[sp+1]:=ac (i:=0)

5: SUBL 2 ; ac:=ac-m[sp+3](i-N)

JZER 15 ; jump if i==N

JPOS 15 ; jump if (i>N)

LODL 0 ; ac:=m[sp+0] (sum)

ADDL 1 ; ac:=ac + m[sp+1] (sum+i)

STOL 0 ; m[sp+0]:=ac (sum=sum+i)

LOCO 1 ; ac:=1

ADDL 1 ; ac:=ac+m[sp+1](1+i)

STOL 1 ; m[sp+1]:=ac (i=1+i)

JUMP 5 ;

15: LODL 0 ; ac:=m[sp+0] (sum)

INSI 2 ; sp:=sp+2 move stack pointer down 2 places

STOL 1 ; m[sp+1]:=sum(return sum)

RETN ; return to caller

3/ STOD result: ; store ac to result location

4/ adder: LODL 1 ; get 1st arg from stack into ac (data count)

STOD mycnt ; store count at location mycnt:

LODL 2 ; get 2nd arg from stack into ac (data addr)

PSHI ; push indirect 1st data to stack

ADDD myc1: ; add 1 (value at myc1:) to addr in ac

STOD myptr: ; store new addr to location myptr:

5/ MySub: LODL 2 ; ac:=topNum

SUBL 3 ; ac:=ac-bottomNum

PUSH

JNEG neg:

LODL 2 ; ac:=result

POPI ; m[ac]=m[sp]

*sp is at local var (value of subtraction),*

*gan gia tri subtraction into result*

LOCO 0 ; ac:=0

RETN

neg: LODL 2

POPI

LODD cn1: ; ac:=m(x); x:address of cn1 & value=-1

RETN

cn1: -1

6/ Recursion

LOOP: LODD PasCnt ; num of fibs to in PasCnt

JZER DONE: ; no more passes, go to DONE

SUBD c1:

STOD PasCnt: ; --pass remaining

P1: LODD daddr: ; load a pointer to fib arg

PSHI ; push arg for fib on stack

ADDD c1:

STOD daddr ; inc, store pointer for next d[n]

CALL FIB: ; call fib (arg on stack)

INSP 1 ; clear stack on fib return

P2: PUSH ; put return AC(fib(n)) on stack

LODD faddr: ; load a pointer to result f[n]

POPI ; pop result off stack into f[n]

ADDD c1:

STOD faddr: ; inc, store pointer for next f[n]

JUMP LOOP: ; go to top for next pass

FIB: LODL 1 ; fib func loads arg from stack

JZER FIBZER: ; if fib(0) go to FIBZER

SUBD c1: ; dec arg value in AC (arg-1)

;;;;Iterative;;;

STOD LpCnt: ; num of iterations in LpCnt

LODD c0: ; load a 0 into AC

STOD fm2: ; store 0 in fib(n-2)

LODD c1: ; load a 1 into AC

STOD fm1: ; store 1 in fib(n-1)

ITER: LODD LpCnt: ; LpCnt arg-1 iterations needed

JZER RTN: ; when LpCnt==0 go to RTN:

SUBD c1: ; dec arg value in AC (LpCnt-1)

STOD LpCnt: ; store LpCnt for next iteration

LODD fm2: ; arg must be >= 2, fm2 initially fib(0)

ADDD fm1: ; fm1 initially fib(1), AC=fm2+fm1

STOD tmp: ; store AC to tmp:

LODD fm1: ; now load AC with fib(n-1)

STOD fm2: ; replace old fib(n-2) with AC

LODD tmp: ; load AC with tmp: becomes fib(n-1)

STOD fm1: ; store AC as next fib(n-1) to fm1

JUMP ITER:

RTN: LODD tmp: ; load AC with tmp: final result

RETN

;;;;Recursion;;;

PUSH ; pushing (arg-1)

CALL FIB: ; fib(arg-1)

PUSH

LODL 1 ; load value(arg-1) to AC

SUBD c1: ; arg-1 is decremented

PUSH ; push(arg-2)

CALL FIB: ; call fib(arg-2)

INSP 1 ; remove (arg-2) from stack

ADDL 0 ; ac=f1+fm2

INSP 2 ; remove f1 & (arg-1) from stack

RETN

;;;;

FIBZER: LODD CO:

RETN ; ac=0 for fib(0)

FIBONE: LODD c1:

RETN ; ac=1 for fib(1)

1/ Convert base 10 into base 2-8-16:

Base 2: 119.78125: 119/2↑.78125\*2↓ => 01110111.11001

Base 8: group 3 digits: 001|110|111|.|110|010| => 167.62

Base 16: group 4 digits: 0111|0111.1100|1000| => 77.C8

2’s complement = flip each single bit + 1

|  |  |
| --- | --- |
| 18.687510 = ?2 18/2 = 9 => 0↑  9/2 = 4 => 1↑  4/2 = 2 => 0↑  2/2 = 1 => 0↑  1/2 = 0 => 1↑  => 1810 = 100102 | .687510 .6875\*2=1.375 => 1↓  .375\*2=0.75 => 0↓  .75\*2=1.5 => 1↓  .5\*2=1.0 => 1↓  => .687510 = .10112  => 18.687510 = 10010.10112 |

2/ Convert 2’s complement into base 10: -1: 1111 -2: 1110 -3: 1101 -4: 0100 -5: 0011 -6: 0010 -7: 0001 -8: 1000 +7:0111 +6:0110 +5: 0101 +4: 0100 +3: 0011 +2: 0010 +1: 0001

Sign bit: Positive: 0 Negative: 1

Magnitude: Flip all digits included sign bit to get magnitude

11011011 => ⌐00100100 + 1 = 001001012 = 37 => Negative: -37

Positive: 0110|10012 = 6916 = 6\*16 + 9 = 10510

3/ Convert \*\* into floating number (base 10):

\*\* IBM format: 1 (Sign)|*7 (Exponent)*|24 (Mantissa) bits

Exponent: base **16**, excess **64**. 0 ≤ Mant. no hidden bit < 1

0:+|*100 0011:67*|0101 1100 0000 0000 0000 0000

Exp.=67-64=3=>163=212=> result =212\*(2-2+2-4+2-5+2-6)

= 210 + 28 + 27 + 26 = 1472.0

\*\* IEEE 754 single precision:

1 (Sign)|*8 (Exponent)*|23 (Mantissa) bits

Base **2** excess **127** Hidden bit = 1; = 0 if exponent is all 0’s

0:+|*1000 0110:134*| 101 1100 0000 0000 0000 0000

134-127=7 => 27\*(1+2-1+2-3+2-4+2-5) = 27+ 26+24+23+22 = 220.0

Ex.: Exponent: 0000 0000: 2-126

1 1111 1111 .0110 = …0000 => NAN

1 1111 1111 .0000 = …0000 => -∞

4/ Convert floating number into IEEE 754 single precision:

.812510 .1101 Move the decimal point to L: +, R: -

\* 2 1.F Move 1 => 127 – 1 = 12610

1.6250↓ => Exponent: 011111102

=> 0|*0111 1110*|101 0000 0000 0000 0000

5/ Add 2 IEEE 754 single precision:

Fl.1: **0**|*1000 0010*|101 1100 0000 0000 0000 0000 => 13.75

Fl.2: **0**|*0111 1110*|101 0000 0000 0000 0000 0000 => 0.8125

Exponent Float 1: 27 + 21 = 128 + 2 = 130 - 127 = 3

Float 2: = 126 - 127 = -1

* shift 4 bits of mantissa 2

Mantissa 1: HB1 101 1100 0000 0000 0000 0000

+ Mantissa 1: HB0 000 1101 0000 0000 0000 0000

110 1001 0000 0000 0000 0000

Result: **0**|*1000 0010*|110 1001 0000 0000 0000 0000

HB1

6/ 01100101.00111:

IEEE 754 shifts 6 places, +6 => 01.10010100111

IBM shifts 8 places, +2 => .0110010100111

IEEE 754: 127+6=133: 1000 0101| 100 1010 0111 0000…

IBM : 64+2=66 : 0100 0010|0110 0101 0011 1000…

7/ Most current machines use 32-bit words. Limit addresses to 4GB, hi-end systems use 64 -bit words. Address space = 1.8\*1019 bytes.

8/ Byte ordering: Conventions

- Big endian: least significant byte has highest (numerically largest) address. Little endian: least sig. has lowest address

Ex: 1521310 = 3B6D16 int: 4 bits

|  |  |  |
| --- | --- | --- |
| 6D | => | 00 |
| 3B | 00 |
| 00 | 3B |
| 00 | 6D |

0x100 0x101 0x102 0x103

Big E: 01 | 23 | 45 | 67 |

Little E: 67 | 45 | 23 | 10 |

-String: char s[6]=”18243”. Character “0” has code 0x30. Digit i has code 0x30+i. Final character=0

-Unsigned values: UMin=0, UMax=2W-1 (-*1 b/c of sign bit*)

2’s complement: TMin=-2W-1, TMax=2W-1-1

-Sign Extension: make k copies of sign bit

-IEEE 754:

Least Positive/N normalized: 0/1 0000 0001 0000…0000 = ±2-126 =~ ±1038

Most 0/1 1111 1110 1111…1111 = ±2128 =~ ±1038

Least Pos./Neg. denorm’ed: 0/1 0000 0000 0000…0000 = ±2-149 =~ ±10-45

Zero 0 0000 0000 0000…0000 = 0

Pos/Neg Infinity 0/1 1111 1111 0000…0000 = ±∞

NAN 0/1 1111 1111 ≠0….. ≠0 = NAN

9/ a) base 10 value if it is a signed 2’s complemented int?

1 111 111 110 010 101 => 2+8+32+64+1 => -107

0 000 000 001 001 101 => 1+4+8+64 => +77

1 111 111 110 010 101

0 000 000 001 001 101 (+)

1 111 111 111 100 010 => 1+4+8+16+1 => -30

**Example 1:** Suppose that IEEE-754 32-bit floating-point representation pattern is **0**|*10000000*|110 0000 0000 0000 0000 0000

Sign bit S = 0 ⇒ positive number

E = 1000 0000B = 128D (in normalized form)

Fraction is 1.11B (with an implicit leading 1) = 1 + 2-1 + 2-2 = 1.7510

The number is +1.75 × 2128-127 = +3.510

**Example 2:** Suppose that IEEE-754 32-bit floating-point representation pattern is **1**|*01111110*|100 0000 0000 0000 0000 0000

Sign bit S = 1 ⇒ negative number

E = 0111 1110B = 126D (in normalized form)

Fraction is 1.1B (with an implicit leading 1) = 1 + 2-1 = 1.5D

The number is -1.5 × 2126-127 = -0.75D

**Example 3:** Suppose that IEEE-754 32-bit floating-point representation pattern is **1**|*01111110*|000 0000 0000 0000 0000 0001

Sign bit S = 1 ⇒ negative number

E = 0111 111010 = 12610 (in normalized form)

Fraction is 1.000 0000 0000 0000 0000 0001B (with an implicit leading 1) = 1 + 2-23

The number is -(1 + 2-23) × 2126-127 = -0.500000059604644775390625 (may not be exact in decimal!)

**Example 4 (De-Normalized Form):** Suppose that IEEE-754 32-bit floating-point representation pattern is **1|***00000000*|000 0000 0000 0000 0000 0001

Sign bit S = 1 ⇒ negative number

E = 0 (in de-normalized form)

Fraction is 0.000 0000 0000 0000 0000 0001B (with an implicit leading 0) = 2-23

The number is -2-23 × 2-126 = -2×-149 ≈ -1.4×10-45

**Example 5:** -176.375 the sign is negative. 176.375=27 + 25 + 24 + 2-2 + 2-3

Move 7 bits to the L, => 10110000.0112 = 1.01100000112\*27

Calculate exponent = 127+7=134 = 100001102

Result: -176.375=1|10000110|011 0000 0110 0000 0000 00002

**Example 6:** 107=

=> 107.21875=1101011.001112 => shift left 6 places

=> 1.101011001112\*26 Exponent = 127+6=133=

Result in IEEE754: 107.21875=0|10000101|1010110011100000…2

=> 107.21875=1101011.001112 => shift left 8 places => +2

=> 0.01101011001112 => exponent = 64+2=66=010000102

Result in IBM: 107.21875=0|1000010

Subref: LODL 1;

PSHI;

LODL 3

PSHI

LODL 1

SUBL 0

INSP 2

RETN

Denormalized floating point …=> Tiny

Memory address space on the Mic1 is the best => 16-bit addressable with 2 to the 12th

EC7A=> FFFF EC7A repeat the sign bit

0 9 5 C 7 6 9 4

Address 0x095C7694=> 0000

12 5C76

Tag = 0x12 Set # = 0x5C76

Tag | Set # | Block offset

16 bits 12 bits 4 bits

4 Hex 3 Hex 1 Hex

Direct-mapped cache with a 16-byte (24 => offset=4) block size and 4K (=212 => set # = 12) sets.

E8|79F|0 miss

E8|79F|4 hit

A4|79F|8 miss

E8|79F|8 miss (conflict)

A4|79F|C miss (conflict

Tavrg seek = 7ms

Tavrg rotation = ½ \* (60 sec/10,000rpm) \* (1000ms/sec) = 3(ms)

Tavrg transfer = (60 sec/10,000rpm)\*(1/1,000sects/track)\*1,000ms/sec = 0.006ms

Taccess = 7 + 3 + .006 = 10.006ms

Taccess = Tavrg seek + Tavrg rotation + Tavrg transfer

Tavrg rotation = ½ \* 1/RPMs \* 60sec/1min

Tavrg transfer = 1/RPM \* 1/(avr # sectors/track) \* 60 sec/1min

Example: Given Rotation rate =7200rpm, avr seek time = 0ms, avg # sector/track = 400.

Derived:

Tavrg rotation = ½ \* (60secs/7,200rpm)\*1,000ms/sec=4ms.

Tavrg transfer = 60/7,200rpm\*1/400 sects/track \* 1,000ms/sec = 0.02ms.

Taccess = 9ms+4ms+0.02ms

Begins at line 100

100: mar := sp; rd;

101: rd;

102: a := mbr;

103: b := a;

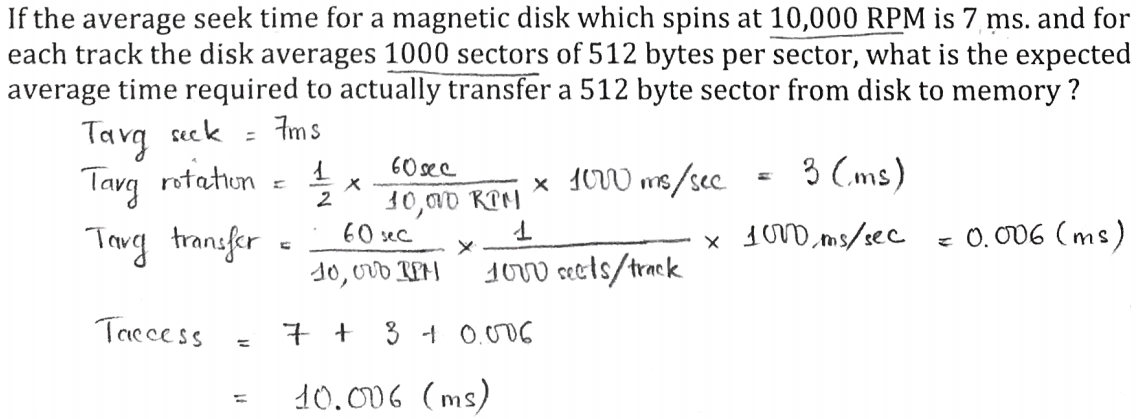
104: b := a + b;

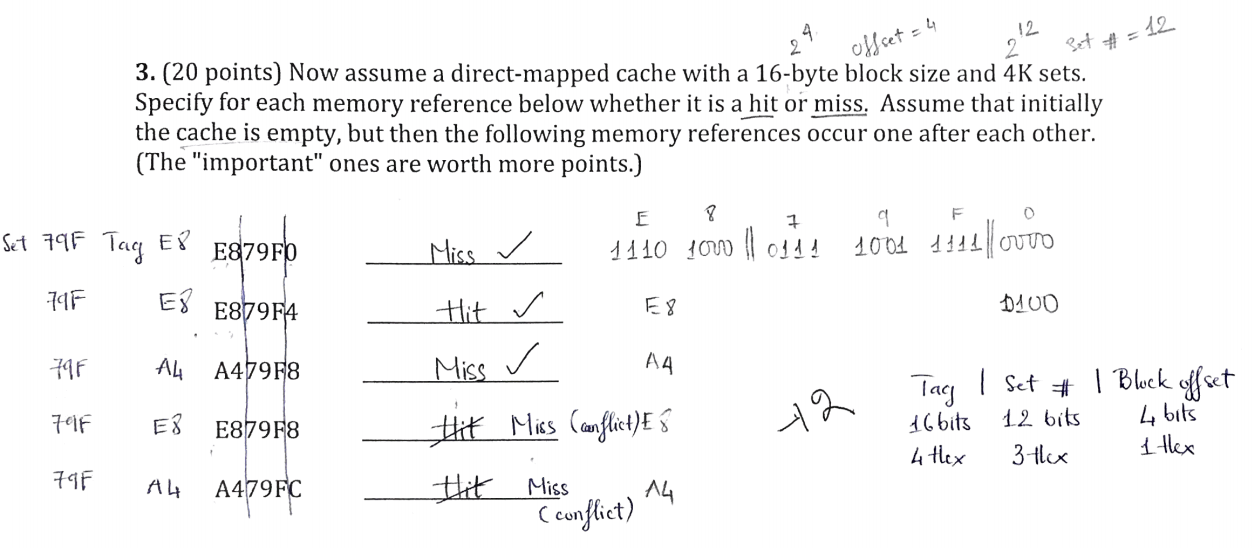
105: b := b + b;

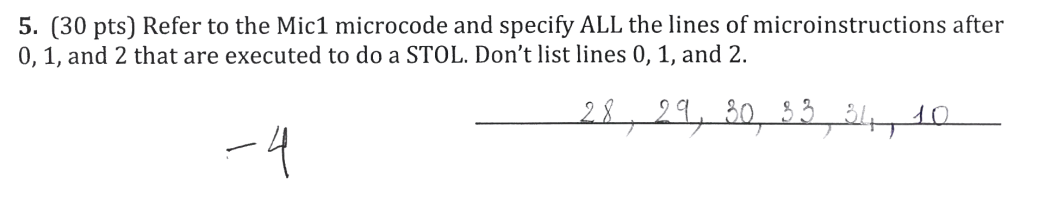
106: b := a + b;

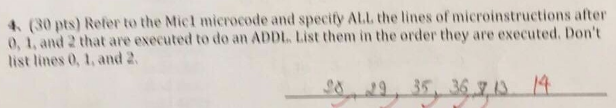
107: mbr := b; wr;

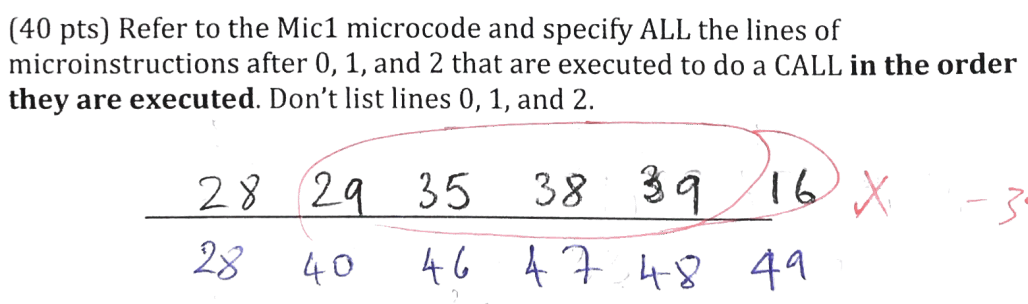
108: wr; goto 0;

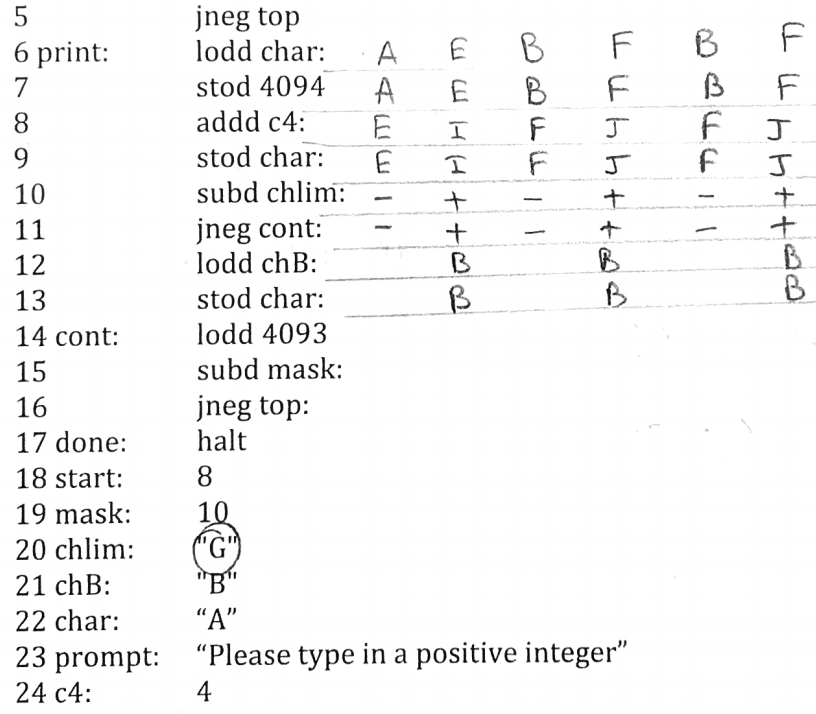


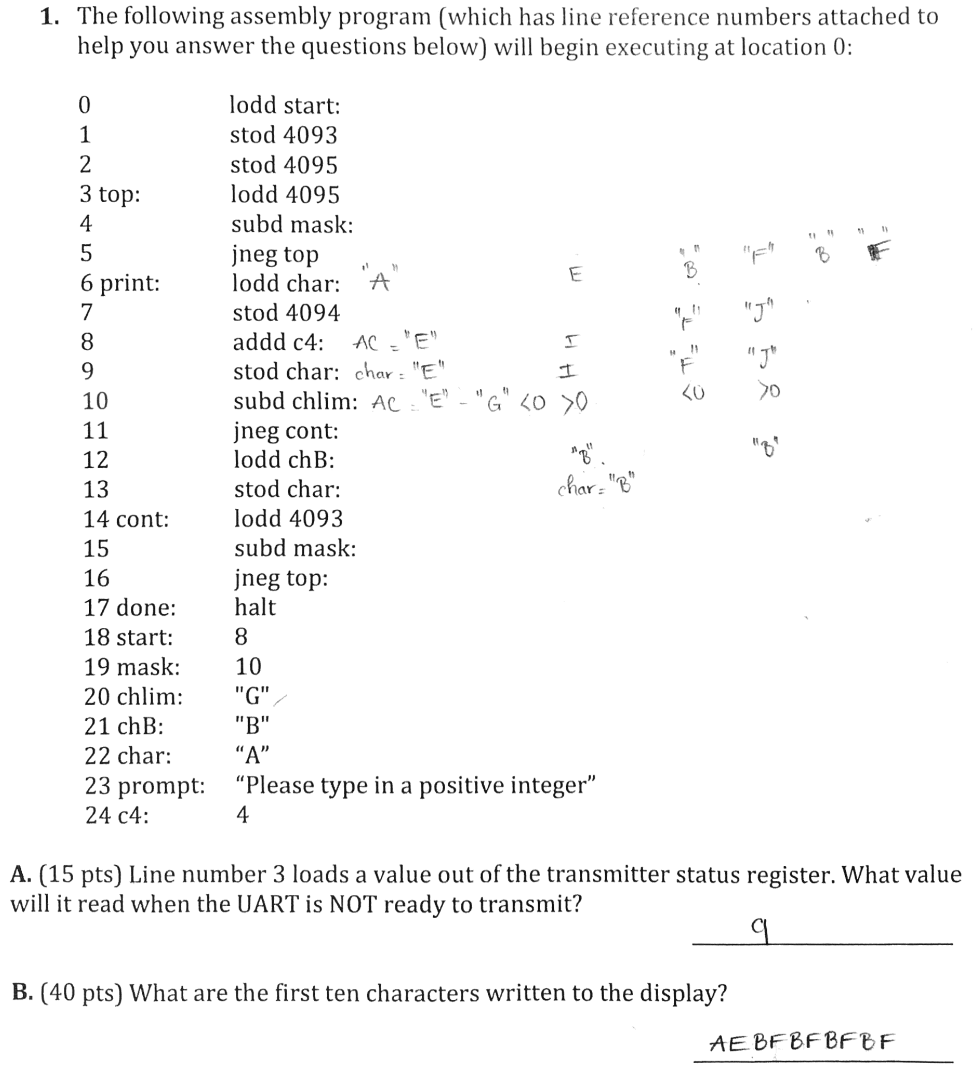


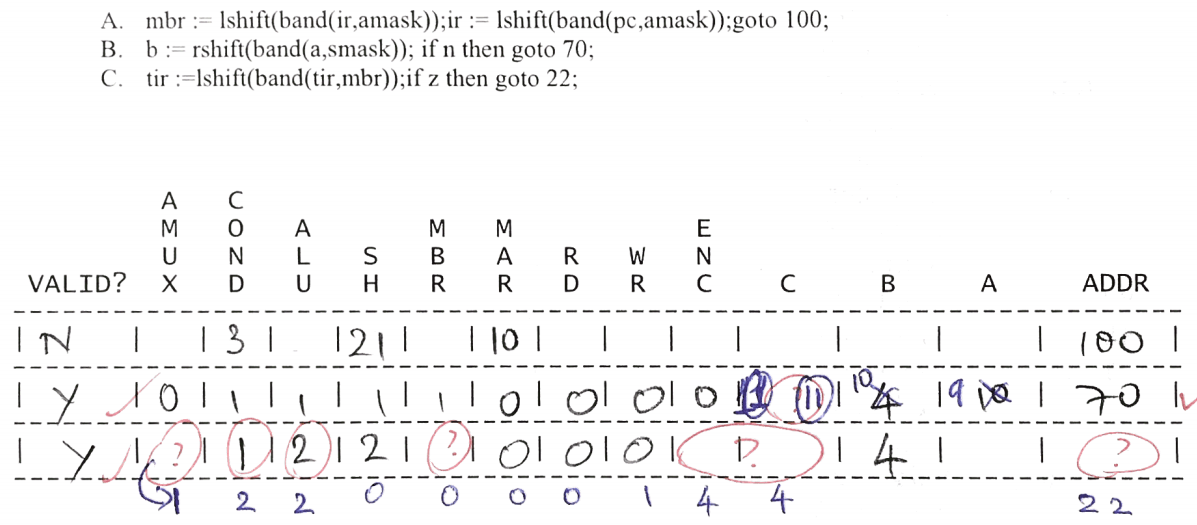


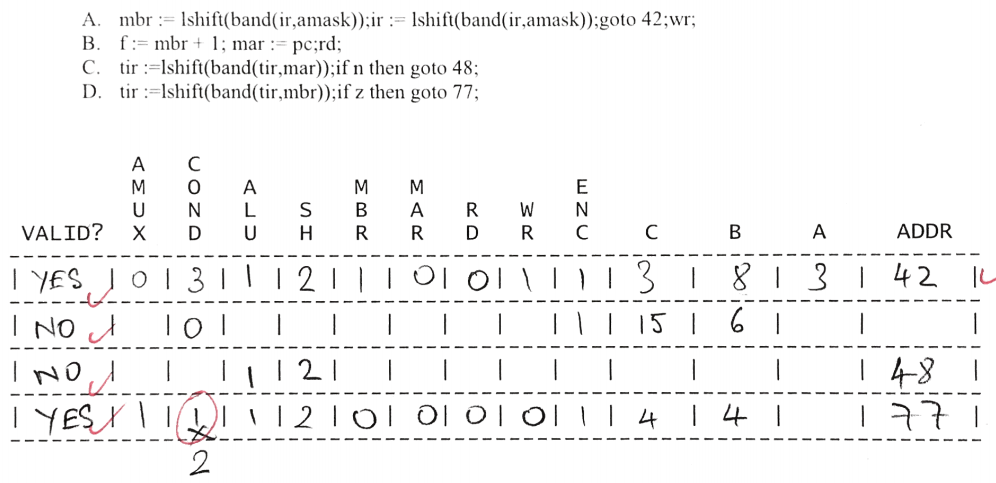


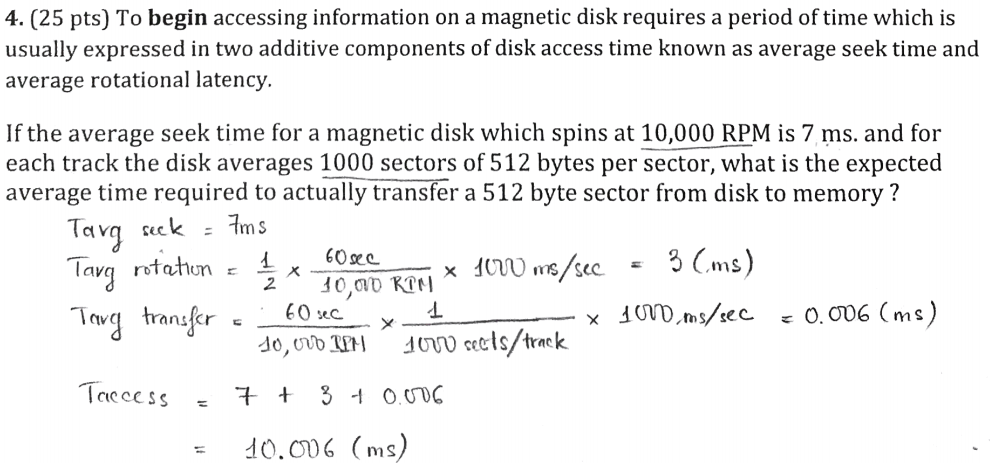












\*\* IBM format: 1 (Sign)|*7 (Exponent)*|24 (Mantissa) bits

Exponent: base **16**, excess **64**. 0 ≤ Mant. no hidden bit < 1

0:+|*100 0011:67*|0101 1100 0000 0000 0000 0000

Exp.=67-64=3=>163=212=> result =212\*(2-2+2-4+2-5+2-6)

= 210 + 28 + 27 + 26 = 1472.0

\*\* IEEE 754 single precision:

1 (Sign)|*8 (Exponent)*|23 (Mantissa) bits

Base **2** excess **127** Hidden bit = 1; = 0 if exponent is all 0’s

0:+|*1000 0110:134*| 101 1100 0000 0000 0000 0000

134-127=7 => 27\*(1+2-1+2-3+2-4+2-5) = 27+ 26+24+23+22 = 220.0

lodl 1; get address of Num1

pshi ; push value of num onto stack

lodl 3; get address of num2

pshi ; push value of num2 onto stack

lodl 0; AC = num2;

addl 1; AC=num2 + num1;

insp 2; clean up stack for two values pushes

retn