CSE 321a: Computer Organization (I)
Third Year, Computer & Systems Engineering

Solution to Assignment #3

1. **Consider an 8-bit stack machine with the following characteristics: …**
   (a) Calculate the maximum number of opcodes that can be included in this instruction set? Justify your answer and suggest another way for encoding the opcodes in order to maximize their number without changing the instruction format.

   Most significant bit (MSB) of the opcode is used to distinguish between zero-operand and one-operand instructions.
   - All zero-operand opcodes have the same value for their MSB. Maximum number of zero-operand opcodes = $2^7 = 128$
   - All one-operand opcodes have the same value for their MSB. Maximum number of zero-operand opcodes = $2^5 = 32$

   Maximum number of opcodes = 128 + 32 = 160

   The maximum possible number of opcodes can be achieved by minimizing the number of one-operand opcodes! In other words, if only **one** 6-bit combination (e.g., 000000) is used to encode a one-operand opcode, then the remaining 63 6-bit combinations (e.g., 000001 → 111111) can be used to encode zero-operand opcodes. This makes it possible to encode 63*4 zero-operand opcodes (e.g., 000001XX → 111111XX).
   Maximum possible number of opcodes in this instruction set = 252 + 1 = 253

   (b) Translate the following C-code snippet into the assembly language of this machine: …

   (*) Solution #1:

```
PUSH R1(2)
SKIPZ #13
PUSH R1(3)
NEGATE
PUSH R1(2)
MULTIPLY
PUSH #50
ADD
POP R1(3)
SKIP #5
PUSH R1(3)
NEGATE
POP R1(3)
```

   (*) Solution #2 (better):

```
PUSH R1(2)
SKIPZ #11
PUSH R1(3)
```

```
NEGATE
PUSH R1(2)
MULTIPLY
PUSH #50
ADD
SKIP #3
PUSH R1(3)
NEGATE
POP R1(3)
```

- **Note: there are many other correct solutions!! For instance, it is possible to push the value 50 to the stack before doing the negation and multiplication. In other words, "`PUSH #50`" can be moved to be right after "SKIPZ #xx". Also, the second "`PUSH R1(2)`" can be moved to be right before the first "`PUSH R1(3)`".**

(c) Fill up the following table with the rest of the machine program:

(*) Solution #1:

Address (Hexadecimal)	Contents (Hexadecimal)
1D	5F
1E	09
1F	78
20	0D
21	5F
22	0D
23	43
24	5F
25	09
26	73
27	5C
28	32
29	4B
2A	63
2B	0D
2C	74
2D	05
2E	5F
2F	0D

30	43
31	63
32	0D

(*) Solution #2:

Address (Hexadecimal)	Contents (Hexadecimal)
1D	5F
1E	09
1F	78
20	0B
21	5F
22	0D
23	43
24	5F
25	09
26	73
27	5C
28	32
29	4B
2A	74
2B	03
2C	5F
2D	0D
2E	43
2F	63
30	0D

(d) Show, using the table below, the execution trace of the machine program (from part (b)) …

(*) Solution #1:

Instruction	PC	SP	R1	Memory Locations			
				72	73	CE	CF
Initially	1D	D0	70	06	85	BE	EF

5F09	1F	CF	70	06	85	BE	06
780D	21	D0	70	06	85	BE	06
5F0D	23	CF	70	06	85	BE	85
43	24		70	06	85	BE	05
5F09	26	CE	70	06	85	06	05
73	27	CF	70	06	85	06	1E
5C32	29	CE	70	06	85	32	1E
4B	2A	CF	70	06	85	32	50
630D	2C	D0	70	06	50	32	50
7405	33	D0	70	06	50	32	50

(*) Solution #2:

Instruction	PC	SP	R1	Memory Locations			
				72	73	CE	CF
Initially	1D	D0	70	06	85	BE	EF
5F09	1F	CF	70	06	85	BE	06
780B	21	D0	70	06	85	BE	06
5F0D	23	CF	70	06	85	BE	85
43	24		70	06	85	BE	05
5F09	26	CE	70	06	85	06	05
73	27	CF	70	06	85	06	1E
5C32	29	CE	70	06	85	32	1E
4B	2A	CF	70	06	85	32	50
7403	2F	CF	70	06	85	32	50
630D	31	D0	70	06	50	32	50

2. Describe the sequence of micro-operations (and show the steps) required to execute each of the following instructions:

(a) `POP (2000)`

t_1: MAR ← [SP]
t_2: MBR ← Memory, SP ← [SP] + 1, MAR ← [IR-address]
t_3: Memory ← [MBR]

(b) `BSA @(-100)` `/* Branch and save address */`

t_1: MAR ← [PC] + [IR-address], MBR ← [PC]

t_2: Memory ← [MBR], PC ← [MAR] + 1 (Assumption: MAR is connected to ALU input *)

(c) `MOVE (700)(R2),500(R1)-  /* 1st operand is destination */`

(*) Solution #1: post-indexing is transformed to regular indexing during the indirect cycle!

t_1: MAR ← [IR-address2] + [R1]
t_2: MBR ← Memory, R1← [R1] – 1
t_3: MAR ← [IR-address1] + [R2]
t_4: Memory ← [MBR]

(*) Solution #2: post-indexing is handled during the execute cycle!

t_1: MAR ← [IR-address1]
t_2: MBR ← Memory
t_3: IR-address1 ← [MBR], MAR ← [IR-address2] + [R1]
t_4: MBR ← Memory, R1← [R1] – 1
t_5: MAR ← [IR-address1] + [R2]
t_6: Memory ← [MBR]