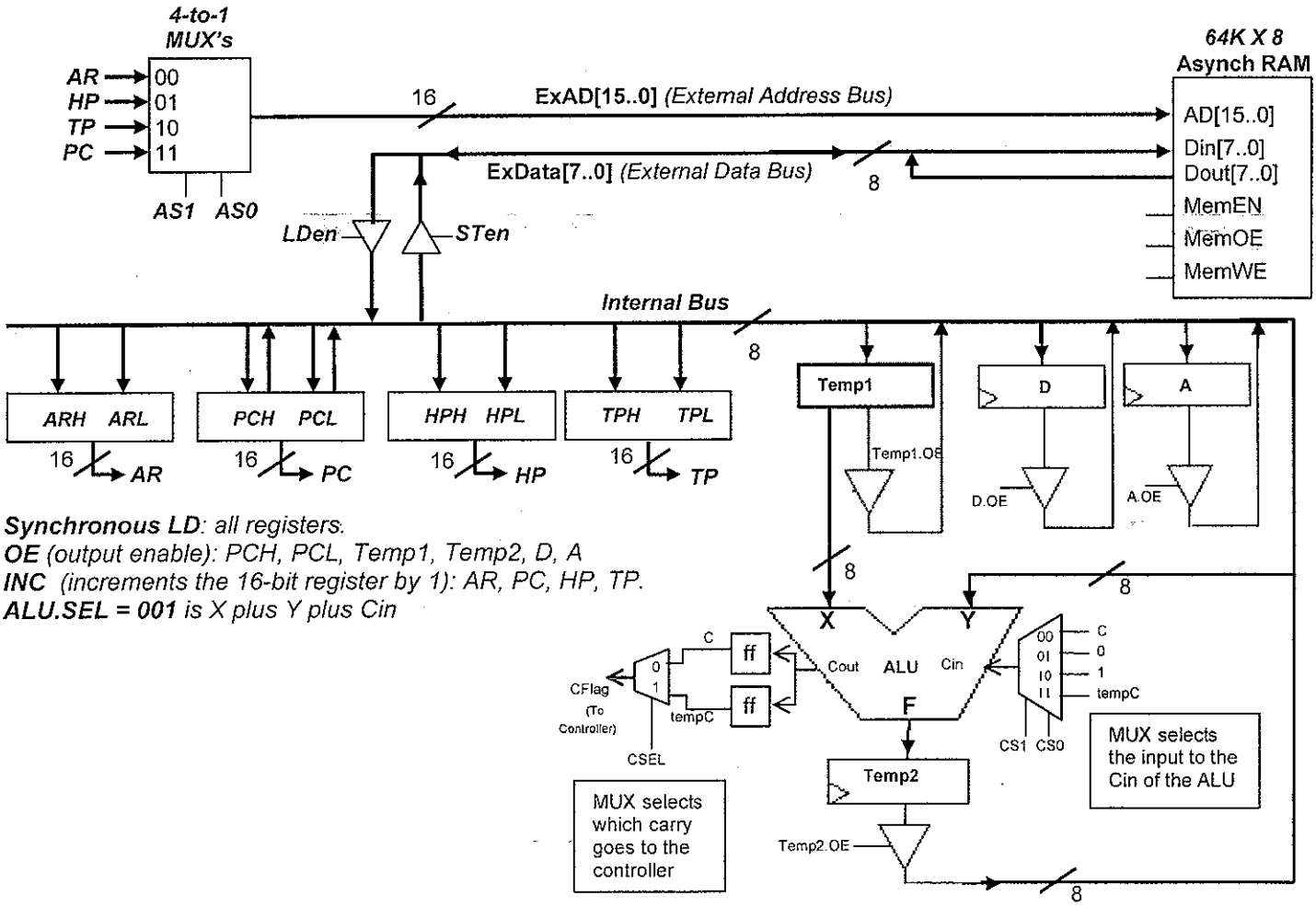
 Name Solution

Small8-related problem: to be used for Problems 1, and 2.

Shown below is a modified architecture of the Small8 computer. Unless specified otherwise, assume that all the components are the same as you had them in your mini-project.

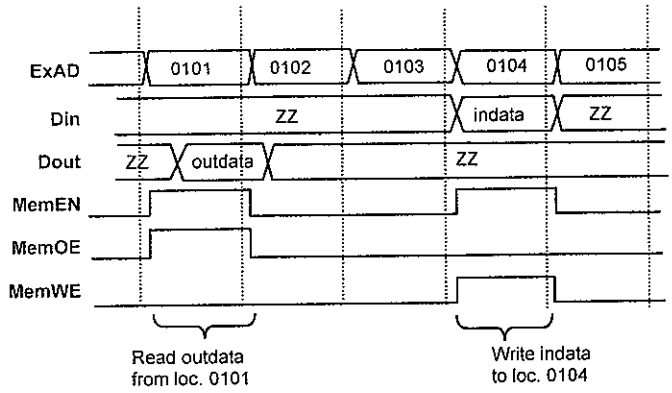
- HP is a 16-bit "Head" Pointer Register (HPH and HPL), containing the head of a queue. The functions of a queue will be described on the next page.
- TP is a 16-bit "Tail" Pointer Register (TPH and TPL), containing the tail of a queue. The functions of a queue will be described on the next page.
- The outputs of AR, HP, TP, and PC are connected to the External Address Bus through a set of 4-to-1 MUX's.
- The RAM is a 64K X 8 **asynchronous** RAM. The timing requirements of its read and write operations are given in the timing diagram on the next page.
- There is a new flag register called tempC (temporary carry).
- Registers Temp1 and Temp2 are connected as shown, both with a syncrhronous **LD** input. Temp1 also has a synchronous **CLEAR** input.



***Synchronous LD**: all registers.*
***OE** (output enable): PCH, PCL, Temp1, Temp2, D, A*
***INC** (increments the 16-bit register by 1): AR, PC, HP, TP.*
***ALU.SEL = 001** is X plus Y plus Cin*

 Name _____

Timing requirements for the read and write operations of the asynchronous RAM:



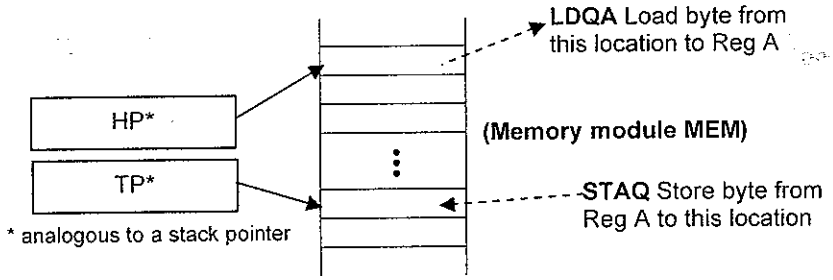
1. You are to implement three new instructions for the Small8 computer by completing the ASM chart on the next page for queue operations: LDQI $addr, LDQA, and STAQ.

18 pts.

A stack is a "first-in, last-out" (FILO) structure. In other words, the first byte stored (pushed) onto the stack is the last byte retrieved (popped), much like a "stack" of trays in a cafeteria.

A queue, on the other hand, is a "first-in, first out" (FIFO) structure. For a queue, the first byte stored is the first to be retrieved, much like the servicing of a line of customers in a cafeteria.

The functions (and Small8 instructions) of a queue are defined as follows:



In more detail, the Small8 instructions for queue operations are defined as follows:

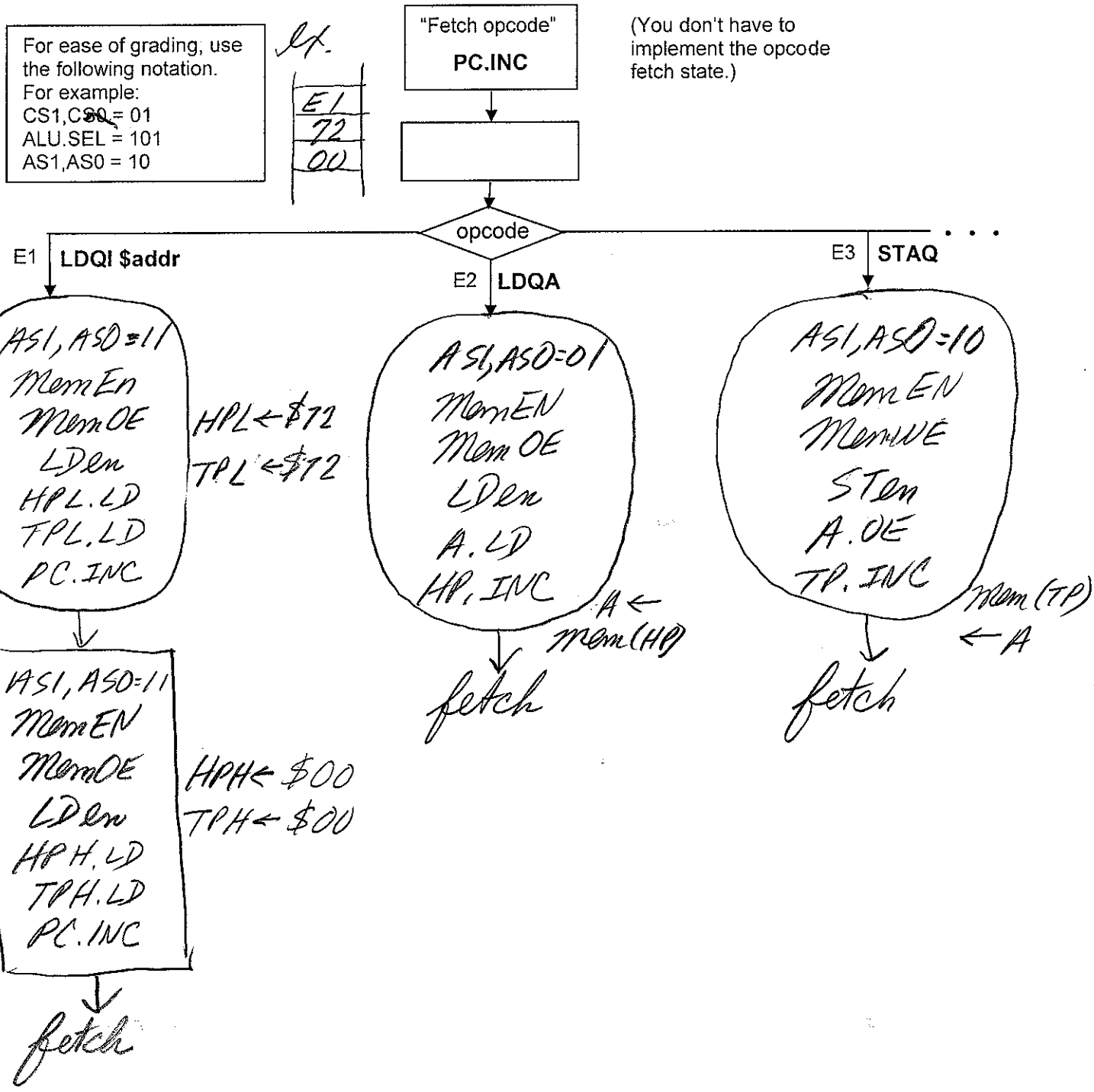
Instruction	Opcode/description	
LDQI $addr	(E1)	Initialize both HP and TP to the 16-bit $addr; (This is a 3-byte instruction.) Example: LDQI $0070 will initialize HP = $0070 and TP = $0070.
LDQA	(E2)	Load the byte from MEM(HP) to RegA; then HP <= HP + 1. (one-byte instruction)
STAQ	(E3)	Store the byte in RegA to MEM(TP); then TailPtr <= TP + 1). (one-byte instruction)

Notes:

- The queue is implemented in MEM (just like the stack in Small8).
- HP is a register containing the memory address (16 bits) of the head of the queue.
- TP is a register containing the memory address (16 bits) of the tail of the queue.
- **You don't have to worry about whether the queue is empty or full.**

1. (continued) Complete the following ASM chart. Note that you **do not** have to complete the opcode fetch state.

For maximum credit, use the minimum number of states (including the use of conditional outputs). For maximum partial credit, "comment" your ASM chart.



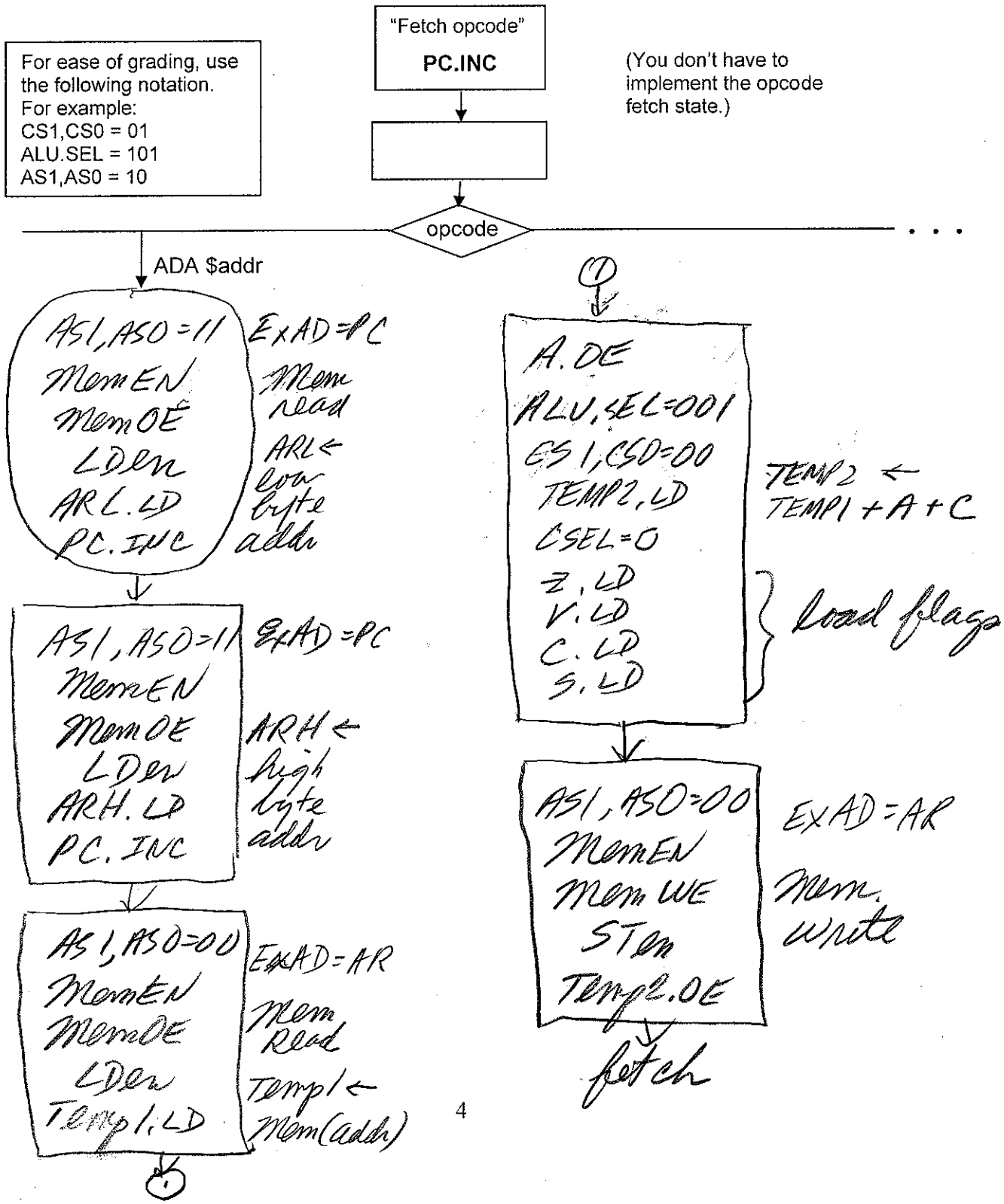
2. You are to implement the ADA instruction by completing the ASM chart.

20 pts.

ADA $addr **Description:** mem($addr) <= mem($addr) + (A) + (C);

- Add the content of Register A to some content in memory (in location $addr)
- For example ADA $0070 will add the content of Register A to memory location $0070.
- Register A should retain its original value.
- All 4 flags are affected.

For maximum credit, use the minimum number of states (including the use of conditional outputs). For maximum partial credit, "comment" your ASM chart.



3. Assembly language program and .mif file

12 pts.

(a) Given the following .mif file, analyze it and produce the corresponding assembly language program. All addresses and contents are in hex. (4 pts)

Address	Content
0000	84
0001	31
0002	F6
0003	0F
0004	00
0005	BC
0006	31
0007	31
0008	B2
0009	0C
000A	00
000B	21
000C	B5
000D	02
000E	00
000F	47

Put the corresponding assembly program here:

LDAI $31
STAA $000F
LDAA $31, X
ORR D
BEQA $000C
ANDR D
BPLA $0002
not an opcode (data)

(b) Assume that we add one instruction (LDAA $0070) at the beginning of the program, what would be the resulting .mif file. Put the answer there. (8 pts)

LDAA $0070
opcode 88

Address	Content
0000	88
0001	70
0002	00
0003	84
0004	31
0005	F6
0006	12
0007	00
0008	BC
0009	31
000A	31
000B	B2
000C	0F
000D	00
000E	21
000F	B5
0010	05
0011	00
0012	47
0013	
0014	

LDAA $0070

4. Small8 Program Execution – at the *instruction* level

18 pts.

Given below is a .mif file containing a **"non-sense"** program involving the **CALL** and **RET** instructions. Based on the Small8 architecture in your mini-project and the instruction set at the end of this exam, fill in the table at the bottom of this page.

(Hint: it would be best to determine the corresponding assembly program first.)

Address	Content	
0000	88	LDAA $00F1
0001	F1	
0002	00	
0003	C8	CALL $0007
0004	07	
0005	00	
0006	C0	RET
0007	EC	STAA 2,X
0008	02	
0009	C0	RET
000A	71	Not opcode (possibly data)
000B	97	
000C	43	

Important Notes:

The opcodes for the instructions can be found at the last page of the exam.

Initial values:

- Memory location $00F1 = 01
- Other initial values are shown below in the table.
- If any initial value is not specified, assume it is 0.

Based on what you know about your Small 8 CPU, analyze the above .mif file and specify the content (in hex) of each of the registers and memory locations. Note that each row in the table below represents the contents **at the end of the execution of each instruction**. All values should be in HEX.

For ease of grading, **leave the box blank** to indicate no change to a register or memory location.

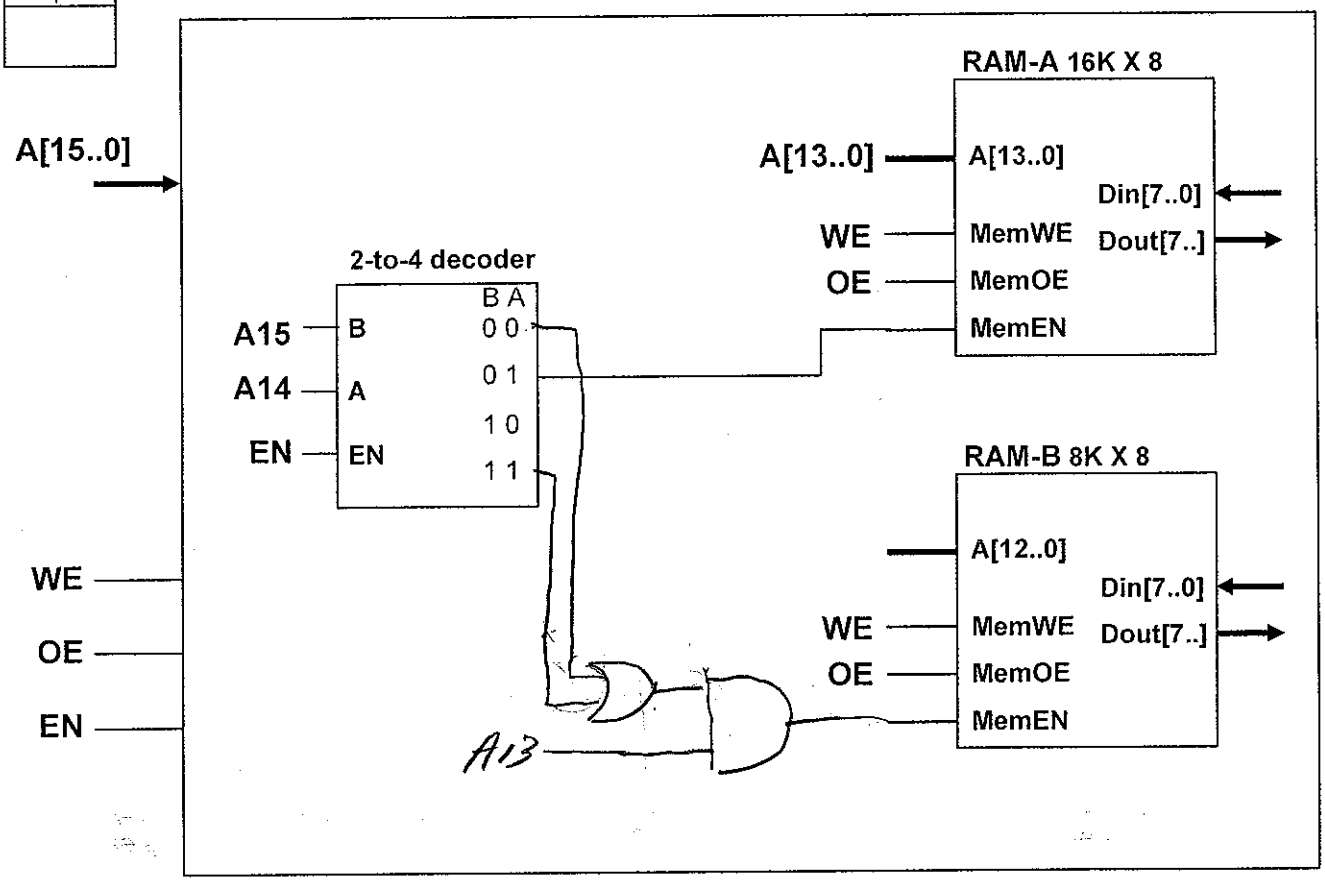
	PC	AR	SP	IX	A	D	Memory Location			
							000A	000B	000C	000D
Initial Values	0000	0000	000A	0009	00	00	00	00	00	00
LDAA $00F1	0003	00F1			01					
CALL $0007	0007		000C					06	00	
STAA 2,X	0009	000B						01		
RET	0001		000A							
STAR D (F1)	0002					01				

* Fill to the end of the table (i.e., execute only 5 instructions).

Name _____

12 pts.

5. Address decoding.



(a) With the above connections, what is the address range of RAM-A (i.e., first and last addresses)?

First address: $4000 **(in hex)**

Last address: $7FFF **(in hex)**

For credit, show work here:

A15 A14 A13 A12 . . . . A3 A2 A1 A0
0 1 0 0 . . . 0 0 0 0
⋮
0 1 1 1 . . . 1 1 1 1

(c) Let's assume we change the connections as follows: A15 to A of the decoder and A14 to B of the decoder. Now, what is the address range of RAM-A?

First address: $8000 **(in hex)**

Last address: $BFFF **(in hex)**

For credit, show work here:

A15 A14 A13 A12 . . . A3 A2 A1 A0
1 0 0 0 . . . 0 0 0 0
⋮
1 0 1 1 . . . 1 1 1 1

(d) **Back to the original connections** as shown in the above figure (i.e., A15 to B and A14 to A), make **ALL** the appropriate connections to make the RAM-B components to have the following range of addresses. For maximum points, use minimum number of additional gates.

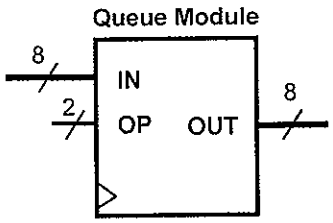
(2000H – 3FFFH) and (E000H – FFFFH); i.e., each location has two addresses.

0010 . . . 000 ; 0011 . . . 111 (A12) ; 1110 . . . 0 ; 1111 . . . 1 (A12) ⇒ A15 A14 can be 00 or 11, A13 must be 1

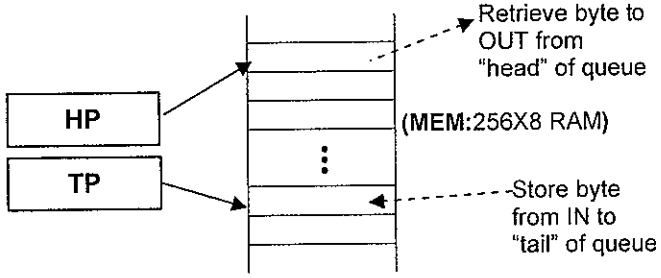
20 pts.

6. ASM design: hardware queue component.

In Problem 1, you designed three instructions to implement and use a queue data structure in the Small8 CPU (LDQI $addr, LDQA, STAQ). In this problem, you will design a stand-alone hardware Queue Module (nothing to do with Small8). To refresh your memory, a queue is a "first-in, first out" (FIFO) structure. For a queue, the first byte stored is the first to be retrieved, much like the servicing of a line of customers in a cafeteria.



(a) Block diagram for Queue Module



(b) Inside of the Queue Module: conceptually

The block diagram the hardware Queue Module is shown above, with two inputs: IN (8-bit input data) and OP (a 2-bit op code). It has one 8-bit output (OUT).

The functions of the Queue Module are determined by the 2-bit op code (OP):

- OP = 00 Do nothing and OUT is high-Z.
- OP = 01 Initialize Head Pointer HP = $00 and Tail Pointer TP= $00 (where $00 is address $00 of a 256X8 bit MEM)
- OP = 10 Retrieve the data byte from the queue "head"; i.e., MEM(HP) is connected to OUT as long as OP = 10; When OP changes, increment the HP register (i.e., HP <= HP + 1).
- OP = 11 Store the data byte from IN to the queue "tail"; i.e., Mem(TP) <= IN; then increment TP (i.e., TP <= TP+ 1);

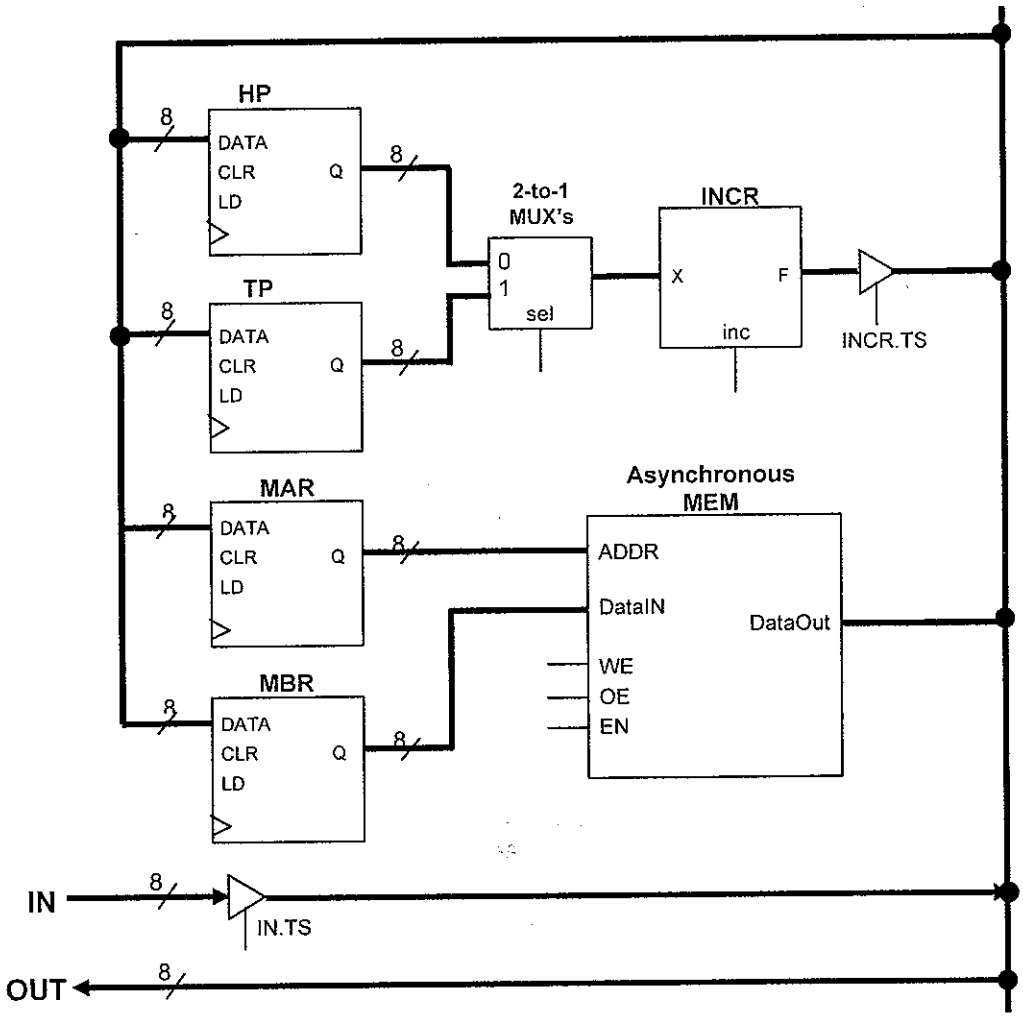
Notes:

The architecture to be used for the Queue Module is given on the next page (again nothing to do with Small8).

- The queue is implemented in a 256 X 8 asynchronous RAM.
- HP is a register containing the memory address (8 bits) of the head of the queue.
- TP is a register containing the memory address (8 bits) of the tail of the queue.
- You don't have to worry about whether the queue is empty or full.

A controller will be used to control the architecture (shown on the next page) to implement the Queue Module.

For this problem, all you have to do is to design this controller (i.e., give me the ASM chart on page 10).



For the following registers: **HP, TP, MAR, MBR**:

- CLR – synchronous clear
- LD – synchronous load

For the **INCR** component:

- If inc = 0, then F <= X
- If inc = 1, then F <= X + 1 (i.e., increment X)

For the asynchronous MEM module:

- ADDR is an 8-bit address bus (i.e., 256 locations)
- The data bus is separated into two buses:
  - DataIN
  - DataOUT
- MEM functions as follows:

EN	OE	WE	Operation	0 = low
0	X	X	DataOut <= High-Z	1 = high
1	1	0	DataOut <= MEM[Addr]	(Memory read)
1	X	1	MEM[Addr] <= DataIn; DataOut <= High-Z	(Memory write)

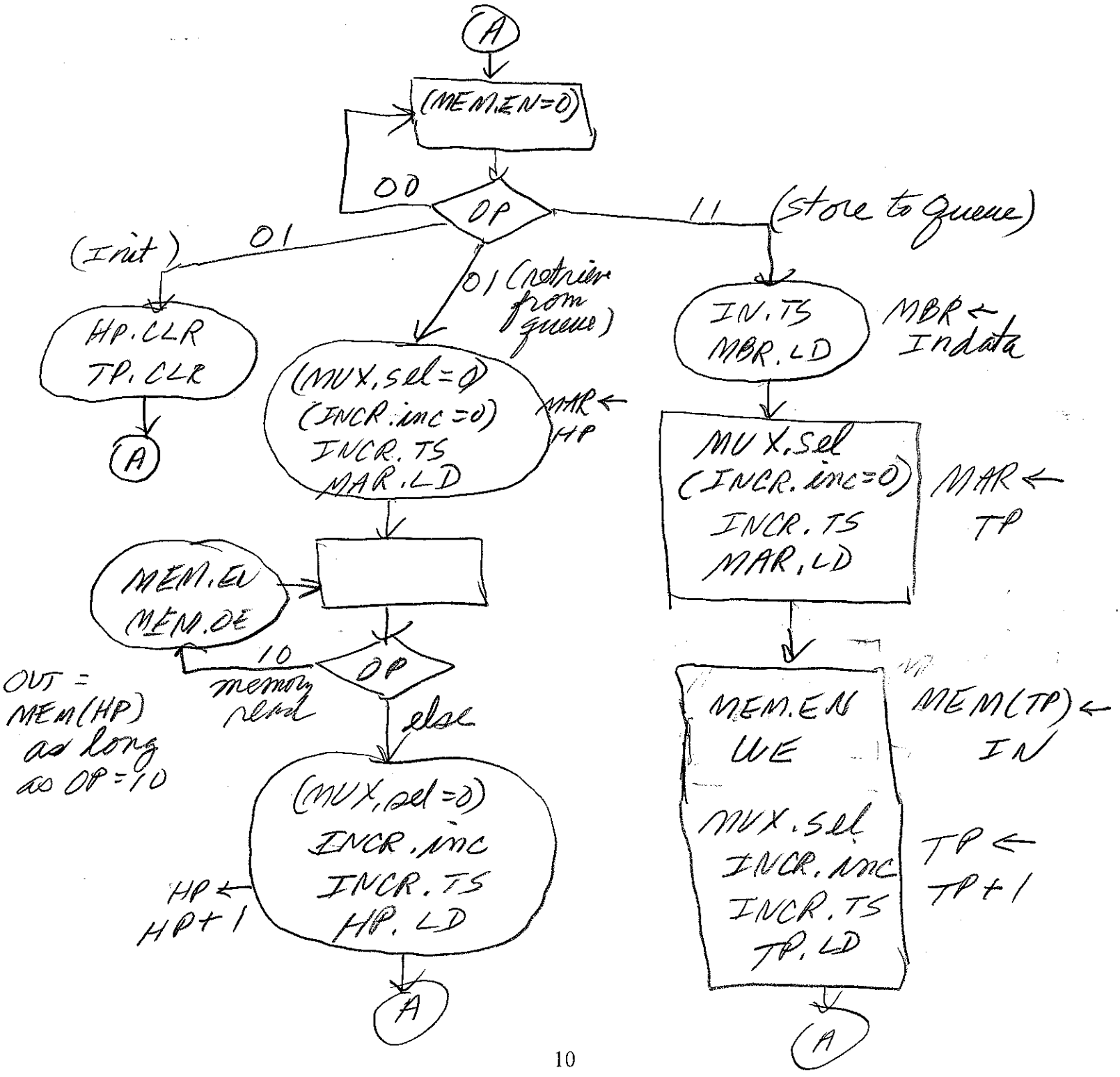
Assume that MEM is asynchronous and memory read and write can be done in 1 clock cycle.

6 (continued) Give the ASM chart necessary to implement the controller for the Queue Module.

- The best answer gets the most points.
- For partial credit, comment the function of the states and conditional outputs.

Hint: Here are the inputs and outputs of this controller.

- **Inputs:** OP, CLOCK
- **Controller outputs:** HP.CLR, HP.LD, TP.CLR, TP.LD, MAR.CLR, MAR.LD, MBR.CLR, MBR.LD, MEM.WE, MEM.OE, MEM.EN, MUX.sel, INCR,inc, INCR.TS, IN.TS



 Name _____

IMPORTANT:

- Please be neat and write (or draw) carefully. If we cannot read it with a reasonable effort, it is assumed wrong.
- As always, the best answer gets the most points.

COVER SHEET:

Problem:	Points:
1 (18 pts)	
2 (20 pts)	
3 (12 pts)	
4 (18 pts)	
5 (12 pts)	
6. (20 pts)	

Total ☐

Re-Grade Information:

INSTRUCTION	OP CODE	C	V	Z	S	DESCRIPTION	SYNTAX
Load Acc (Imm)	84	×	×	✓	✓	A ← mem[PC]	LDAI <data>
Load Acc (Abs)	88	×	×	✓	✓	A ← mem(mem[PC])	LDAA <address>
Load Acc (RR)	81	×	×	✓	✓	A ← (D)	LDAD
Store Acc (Abs)	F6	×	×	×	×	Mem(mem[PC]) ← (A)	STAA <address>
Store Acc (RR)	F1	×	×	×	×	D ← (A)	STAR D
Add with Carry	01	✓	✓	✓	✓	A ← (A) + (D) + C	ADCR D
Subtract with Borrow	11	✓	✓	✓	✓	A ← (A) + not(D) + C	SBCR D
Compare	91	✓	✓	✓	✓	Same as Subtract, but only change Status Flags (A is unchanged)	CMPR D
AND	21	×	×	✓	✓	A ← (A) AND (D)	ANDR D
OR	31	×	×	✓	✓	A ← (A) OR (D)	ORR D
XOR	41	×	×	✓	✓	A ← (A) XOR (D)	XORR D
Shift Left Logical	51	✓	×	✓	✓	C ← (A7), A7 ← A6, …, A0 ← 0	SLRL
Shift Right Logical	61	✓	×	✓	✓	C ← (A0), A0 ← A1, …, A7 ← 0	SRRL
Rotate Left through Carry	52	✓	×	✓	✓	C ← (A7), A7 ← A6, …, A0 ← C	ROLC
Rotate Right through Carry	62	✓	×	✓	✓	C ← (A0), A0 ← A1, …, A7 ← C	RORC
Branch on /C (Inh)	B0	×	×	×	×	if (C=0), PC ← mem[PC] else PC++	BCCA
Branch on C (Inh)	B1	×	×	×	×	if (C=1), PC ← mem[PC] else PC++	BCSA
Branch on Z (Inh)	B2	×	×	×	×	if (Z=1), PC ← mem[PC] else PC++	BEQA
Branch on S (Inh)	B3	×	×	×	×	if (S=1), PC ← mem[PC] else PC++	BMIA
Branch on /Z (Inh)	B4	×	×	×	×	if (Z=0), PC ← mem[PC] else PC++	BNEA
Branch on /S (Inh)	B5	×	×	×	×	if (S=0), PC ← mem[PC] else PC++	BPLA
Branch on /V (Inh)	B6	×	×	×	×	if (V=0), PC ← mem[PC] else PC++	BVCA
Branch on V (Inh)	B7	×	×	×	×	if (V=1), PC ← mem[PC] else PC++	BVSA
Decrement Acc	FB	×	×	✓	✓	A ← (A) - 1	DECA
Increment Acc	FA	×	×	✓	✓	A ← (A) + 1	INCA
Set Carry Flag	F8	✓	×	×	×	C ← 1	SETC
Clear Carry Flag	F9	✓	×	×	×	C ← 0	CLRC

Table 1: Small8 Instruction Set

Addendum to the instruction set

INSTRUCTION	OP CODE	C	V	Z	S	DESCRIPTION	SYNTAX
Subroutines:							
Load SP (Imm)	89	X	X	X	X	SP ← mem[PC+1],mem[PC]	LDSI <data>
Call	C8	X	X	X	X	SP ← (SP) + 1; mem[SP] ← (PC_L); SP ← (SP) + 1; mem[SP] ← (PC_H)	CALL <address>
Return	C0	X	X	X	X	PC_H ← mem[SP]; SP ← (SP) – 1; PC_L ← mem[SP]; SP ← (SP) – 1	RET
Index addressing:							
Load X (Imm)	8A	X	X	X	X	X ← mem[PC+1],mem[PC]	LDXI <data>
Load Acc (Indx)	BC	X	X	X	X	A ← mem[(X) + b]	LDAA b,X
Store Acc (Indx)	EC	X	X	X	X	mem[(X) + b] ← (A)	STAA b,X
Increment X	FC	X	X	X	X	X ← (X) + 1	INCX
Decrement X	FD	X	X	X	X	X ← (X) – 1	DECX