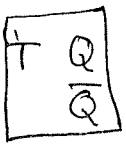
ORG FALL 85 TEST 1 'μ-op'ed by _____

1. FETCH STATE of Ch. 5 computer: Show it all: 'controls', μ-op's and explicitly show where the state changes go to (and when).

2. IEff (Input-Enable flip-flop): When E=0 the IEff holds its current value. Otherwise, when E=1 the input line I is 'loaded' into the IEff. In each part below, use the given flip-flop and NO MORE THAN 3 GATES to realize an IE ff.

A.

B.



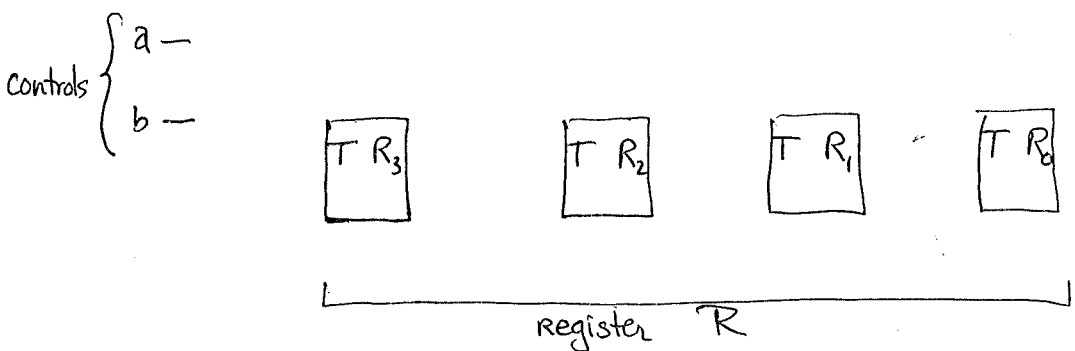
3. X & Y are address in memory for the Ch5 computer. The contents M[X] & M[Y] contained the address for the operands a & b respectively.
   A) Write down a sequence of assembly statements (i.e. machine level) which will do "b = a+b".

   C) How many clock cycles does it take for your program in (A) to run?

   D) The clock on the ch5 computer runs at 2MHz. How much time (in micro-secs (μsec)) [not cycles] does it take for the number of cycles in (C)?

4. Suppose $X = 987_H$ and $Y = 654_H$ in the above problem. Write down μ-op's which when done in sequence will do the same thing (namely $M[M[Y]] \leftarrow M[M[X]] + M[M[Y]]$) using a minimal no. of clock cycles.

5. Using the 4 T flip-flops below realize the register R with controls a & b so that the μ-ops a: $R \leftarrow R+2$, b: $R \leftarrow \overline{R}$ and if (a=0 and b=0) then R holds, all work.



6. FIFO Queue/Circular buffer: 128 word circular buffer is added to the Ch.5 computer. New registers FRONT (12-bits), REAR (12-bits) COUNT (8-bits) and 1-bit flags UE (underflow error) and VE (overflow error) are added. The buffer is in memory addresses $F00_H$ through $F7F_H$. Two new instruction are added:

Inbuff (op code r): ~~AC←1~~ M[REAR] ← AC, INC REAR, INC COUNT
Outbuff (op codes): AC ← M[FRONT], INC FRONT, DEC COUNT
~~over~~ however both pointers wrap around ($F00_H \leq$ FRONT, REAR $\leq F7F_H$)
and the flags UE, VE are set on their occurance
[if "Pop" from empty buff ~~then~~ then UE ← 1 and
if "Push" to full buff then VE ← 1]

YOUR JOB is to write the execution cycle (with controls) for both instructions

A. Inbuff

B. Outbuff

7. The ch.5 computer when faced with a machine instruction of the form [x111|xxxx|xxxx|xxxx] "knows" it is either a register reference (12 possible) or I/O (6 possible) instruction.

A. What is the smallest bit width needed to know which reg refer I/O instruction? (of course the ch 5 computer uses all 13 other bits, but we want the smallest number of bits it could be encoded into.)

B. If we used your answer in (A), how many more such instructions could we "fit" using the same bit field?

C. We now have some "spare bits" i.e. not used by q_7 nor determining which reg ref or I/O instruction. What is the width of this field?

D. Lets us use the field in C to be a signed constant (in 2's comp) which (in one of unused op-codes in (B)) is loaded into the AC. write the μ-op which loads the AC after a ch.5 fetch. [Assume the field in C is the rightmost bits]

E. What is the range of values that the AC could have after such a "load immediate" instruction?

8. In the table below; either write no and say why or write yes see the examples below. Notation & such as in the ch 5 computer

	AC ← AC − M	MBR ← AC	EAC ← AC + AC	MAR ← M(bottom 12 bits)
could be μ-op?				
could be machine level instruction?				

Examples: Consider AC ← AC − 1. Although it is neither an assembly instruction nor a μ-op for the ch.5 computer it could be both so we should answer yes & yes. On the otherhand SC ← 1 could only be a μ-op (why not ass?) and AC ← M[address part of M[X]] could only be an ass. instruction (why not μ-op?)

ORG test 2 13 Nov 85 by _____

1-4 10 pts each 5-8 ; 15 pts each Good Luck!

1. In an 8-bit machine $3C_{16}$ is subtracted from BD_{16}. Give the contents of the flags C, S, V, Z and PE (parity even) after this operation.

2. AB. Formulate a mapping process from (macro) op-codes to (μ) control memory addresses given that μ-memory has 1K words, op-codes are 6 bits wide and 4 control words are needed for each macro-op

C. Illustrate your map with the op-code 42_{10}, give your address in binary.

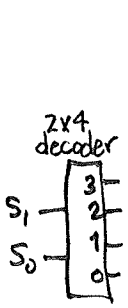
D. What is the size of the CAR?

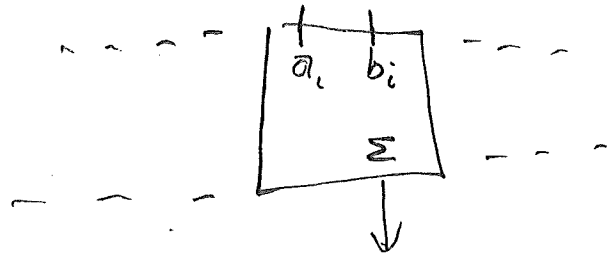
3. AB. The text divides all interrupts into 3 classes. Give these 3 classes and an example of each.

C. What is an interrupt vector and how is it used?

4. Design a circuit for ALU which yields (Actually we need only one "stage" of the logic which is "fed" into a parallel adder.) The decoder will ease the pain

S_1S_0	$C_{in}=0$	$C_{in}=1$
00	A	A+1
01	A+B	A+B+1
10	$A+\overline{B}$	$A+\overline{B}+1$
11	B-1	B





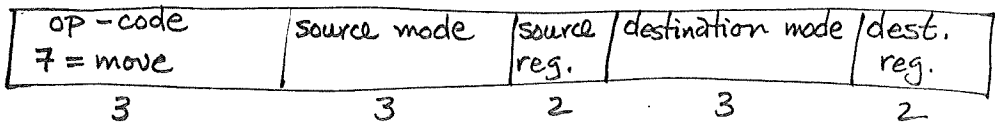
5 (μ-code) The Jazzed Up Chapter 8 computer is improved to include SP (stack pointer register which points to TOS and "grows" toward low memory) and the μ-code format is expanded to include F4 (see →) Op codes 9 and 12 are for CALL and RETURN. Write these in the Ch.8 style, give ORG and worry about effective addresses

A CALL

B RETURN

F4	
0	NOP
1	SPTAR
2	INCSP
3	DECSP
4	BRTSP
5	PCTSP
6	SPTPC
7	———

6. (addressing modes) The test 2 computer is a 13-bit machine with four 13-bit registers (cleverly named) R0, R1, R2 & R3 The general format for an instruction is



op-code '7' is for "move" or more correctly "copy". The "modes" are as follows note for mode = 0 the reg. field doesn't pick a reg but the reg field picks one of the "ext. modes"

mode	
0	memory reference
1	reg direct
2	reg indirect
3	index
4	pre-decrement
5	post-increment

reg (memory reference)	
0	immediate
1	direct
2	indirect
3	relative

Given the instruction at M[0] is the one given below and M[1] contains 'x', M[2] contains 'y' (where addition constants and/or addresses are found) Write the effect of each instruction "á la" M[M[R0+x+M[y]]] ← R2+y"

	OP	SM	SR	DM	DR
A	7	0	0	1	3
B	7	2	2	0	1
C	7	0	3	4	1
D	7	5	0	0	2
E	7	3	2	3	3

(μ-ops)

7.1 This version of the chapter 5 computer has both an SP (points to TOS, grows toward low memory) and an FP (a "frame pointer"). The LINK ass-instruction reserves some stack space (for subroutine variables). The amount of space is given by the address field (it's a memory reference instruction) It does PUSH FP, FP←SP, SP←SP+constant. UNLINK undoes this (but it doesn't know the "constant" (nor does it need it)) Write the execution cycles for these with controls given they have "op-codes" r & s respectively. (Additional Adders are availible)

A. LINK

B UNLINK

8. (parameter passing via the stack) [SP points to TOS, grows toward low memory]

A compiler would generate the code

```
push 10
push 4
push 1
call P
```

for a procedure call P(1,4,10) if the procedure body is as in the box.

```
Procedure P (A,B,C: integer);
[Local variables]
 begin
  :
 end
```

A. Suppose integers and addresses fit into 1 word of memory. At the begining of P the variables A,B,C are easy to obtain via using the SP as an index register. What are the offsets from SP for Var A? Var B? Var C?

B. If Procedure P has local variables it will call LINK (Prob 7) to create stack space for them. The variables A,B,C are still easy to obtain but this time via indexing from the FP. What are the offsets from FP for Var A? Var B? Var C?

CDE. This stack use allows for procedures to be called with a different number of parameters each time it is called. i.e. push the variables onto the stack in backwards order (right to left) then push the size of the variables onto the stack (i.e. reals take 2 words) then push the number of variables onto the stack and call the routine. Like writeln(integer, real, integer)

C. what is the offset of the first variable from SP in general?

D. what is the offset of the second variable from SP?

E the offset for 3?