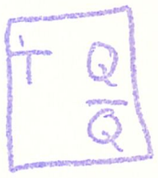
CRG FALL 85 TEST 1 'μ-op'ed by _____

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

[6,1,6,2,2,1,1,0⟶] 80

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.



[0,1,2,5,0,1,0,0,0,1,0] 82

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

AB Write down a sequence of assembly statements (i.e. machine level) which will do "b = a+b".

[5,0,1,2,2,1,2,1,1,4,0] 56

C. How many clock cycles does it take for your program in (AB) 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)?

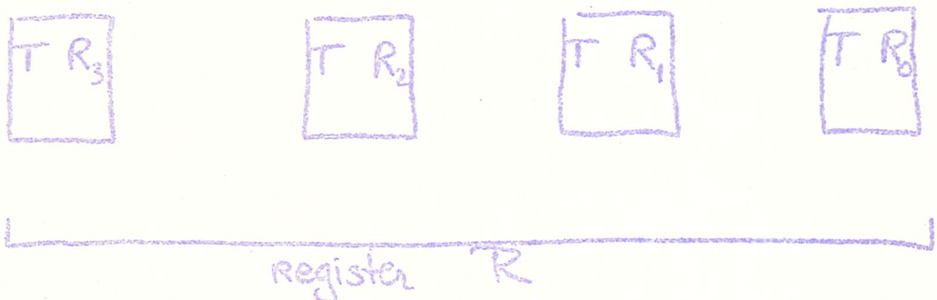
4. Suppose $X = 987_H$ and $Y = 654_H$ in the above problem. Write down μ-op's which when done in sequence will do the same thing (namely $M[M[Y]] \leftarrow M[M[X]] + M[M[Y]]$) using a minimal no. of clock cycles.

[2,6,2,0,2,4,1,0,0,2,0] 67

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

Controls { a —
b —

[7,0,0,6,0,1,1,1,3,0 →] 79



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): M[REAR] ← AC, INC REAR, INC COUNT
Outbuff (op code s): AC ← M[FRONT], INC FRONT, DEC COUNT

~~over~~ however both pointers wrap around ($F00_H \le$ FRONT, REAR $\le F7F_H$)
and the flags UE, VE are set on their occurance
[if "Pop" from empty buff 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

47

[0,0,1,1,1,3,2,1,2,2,1,0,3,0,1,1]

7. The ch.5 computer when faced with a machine instruction of the form [X|111|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 instruction? (of course the ch5 computer uses all 13 other bits, but we want the smallest number of bits it could be encoded into.) reg ref or I/O

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

[0,1,3,2,2,2,1,0,1,1,2,0,0,0,2,2]

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?

54

D. Lets us use the field in C to be a signed constant (in 2's comp) which (in one of the used 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 other hand 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?)

[5,2,6,1,1,1,0,1,1,1,0]

82

CRG 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).

C_0t_0: MAR ← PC, PC ← PC + 1

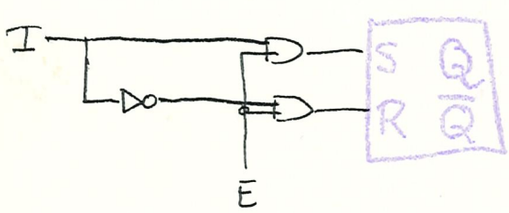
C_0t_1: MBR ← M

C_0t_2: I ← MBR(1), OPR ← MBR(2-4)

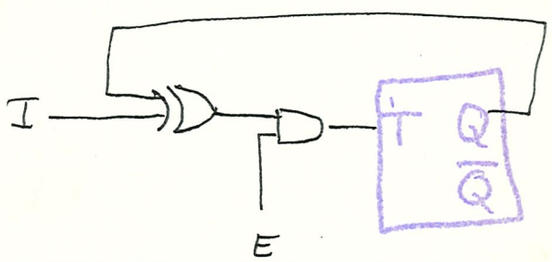
C_0t_3: if $q_7'I$ go indirect ($C_1 \leftarrow 1$ or $R \leftarrow 1$)
else $q_7 I'$ go execute ($C_2 \leftarrow 1$ or $F \leftarrow 1$)

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

AB Write down a sequence of assembly statements (i.e. machine level) which will do "b = a + b".

```
LDA  X  I
ADD  Y  I
STA  Y  I
```

CD. How many clock cycles does it take for your program in (AB) to run?

36

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)?

18

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]] ← M[M[X]] + M[M[Y]]) using a minimal no. of clock cycles.

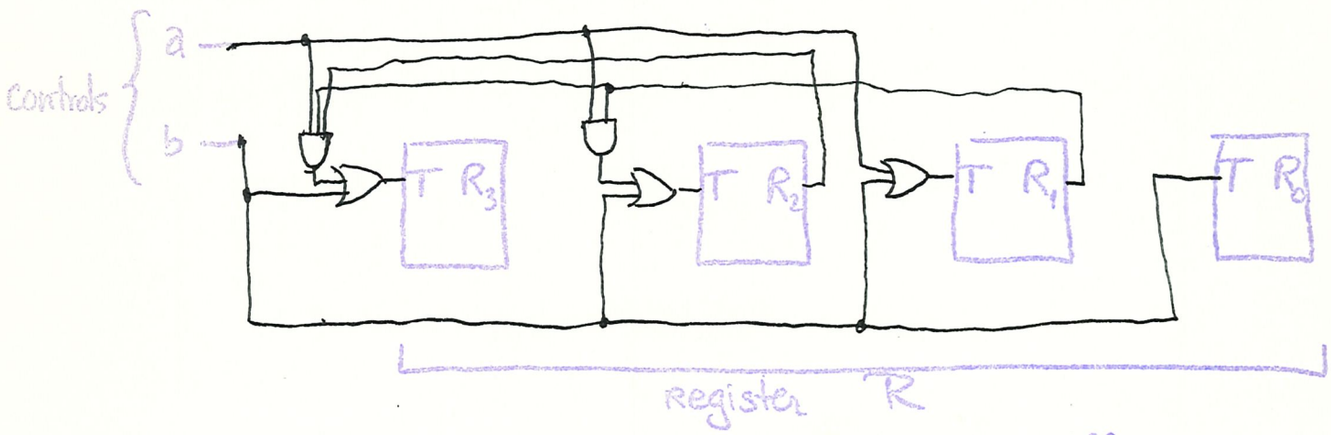
- MAR ← 987
- MBR ← M
- MAR ← MBR(AD)
- MBR ← M, MAR ← 654
- AC ← MBR, MBR ← M
- MAR ← MBR(AD)
- MBR ← M
- MBR ← MBR + AC
- M ← MBR

9 cycles

- MAR ← 987
- MBR ← M, MAR ← 654
- MAR ← MBR(AD), MBR ← M
- MAR ← MBR(AD), MBR ← M
- MBR ← M, AC ← MBR
- MBR ← MBR + AC
- M ← MBR

7 cycles

5. Using the 4 T flip-flops below realize the register R with controls a & b so that the μ-ops a: R ← R+2, b: R ← $\bar{R}$ 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 $F80_H$ through $F7F_4$. Two new instruction are added:

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

rC_2t_0: MBR ← AC, MAR ← REAR, REAR++
COUNT++, if COUNT = 128 then VE ← 1
rC_2t_1: M ← MBR if REAR = F80 then REAR ← F00
rC_2t_2:
rC_2t_3: change states.

B. Outbuff

sC_2t_0: MAR ← FRONT, FRONT++, COUNT--
if count = 0 then UE ← 1
sC_2t_1: MBR ← M, if FRONT = F80 then FRONT ← F00
sC_2t_2: AC ← MBR
sC_2t_3: change states

7. The ch.5 computer when faced with a machine instruction of the form [1111|xxxx|xxxx|xxxx] "knows" it is either a register reference (12 possible) or I/O (6 possible) instruction. $12+6=18$ $2^4<18<2^5$ reg ref or I/O

A. What is the smallest bit width needed to know which 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.) 5

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

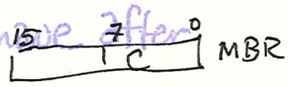
$32-18=14$

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?

$16-3-5=8$

D. Lets us use the field in C to be a signed constant (in 2's comp) which (in one of the used 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]

AC (bottom 8 bits) ← MBR(C) AC (top 8 bits) ← FF if MBR(7)=1, 00 if MBR(7)=0



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

$-2^7 \leq \quad \leq 2^7-1$

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?	no memory reference takes several clock cycles	yes we have seen it often	yes	no memory reference again
could be machine level instruction?	yes (SUB X where X is add of M)	no MBR is invisible at machine level	yes Shift left EAC with zero fill	no MAR like MBR not programable

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?)

[5,2,6,7,1,1,0,1,1,1,0,1,2,7]