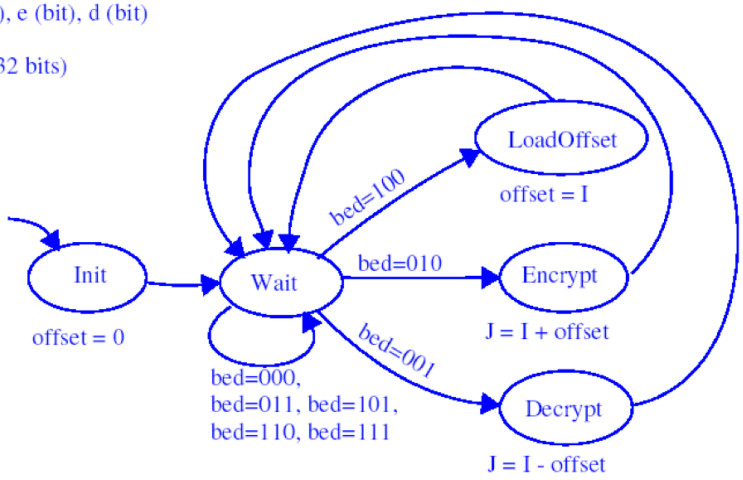
Exercícios do Vahid

5.2)

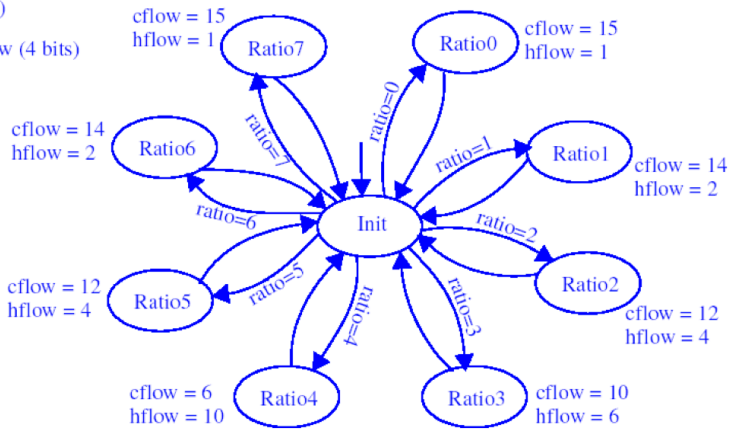
Inputs: I (32 bits), b (bit), e (bit), d (bit)
Outputs: J (32 bits)
Local registers: offset (32 bits)



5.3)

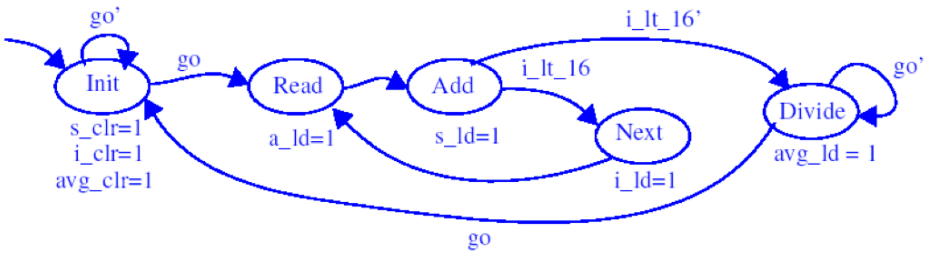
Inputs: ratio (3 bits), on (bit)

Outputs: hflow (r bits), cflow (4 bits)



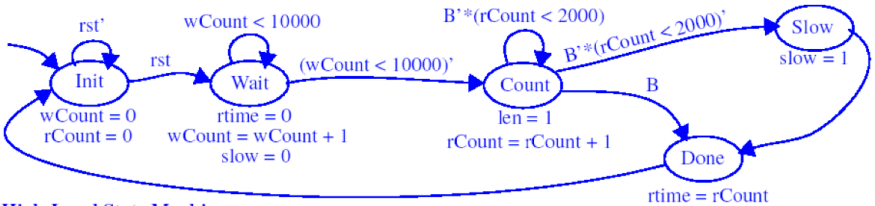
5.11)

Inputs: go, i_lt_16 (bit)
Outputs: s_clr, i_clr, avg_clr, s_ld, i_ld, a_ld (bit)

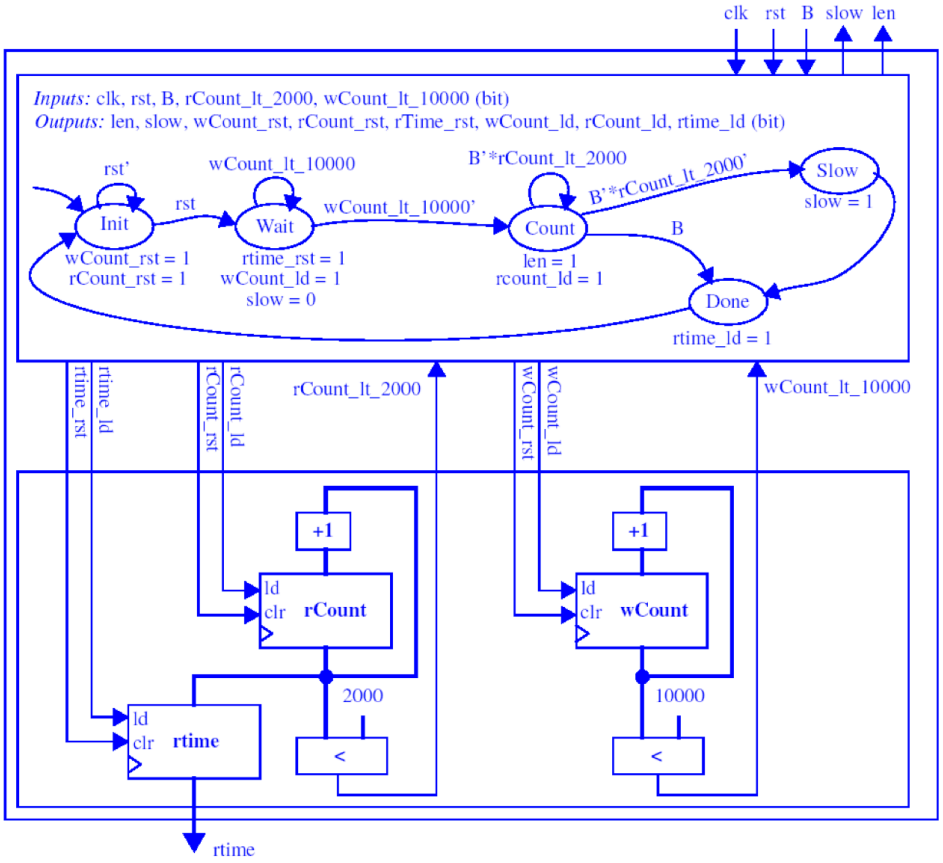


5.12)

Inputs: clk, rst, B (bit)
Outputs: len, slow (bit); rtime (11 bits)
Local Registers: wCount (14 bits); rCount (11 bits)



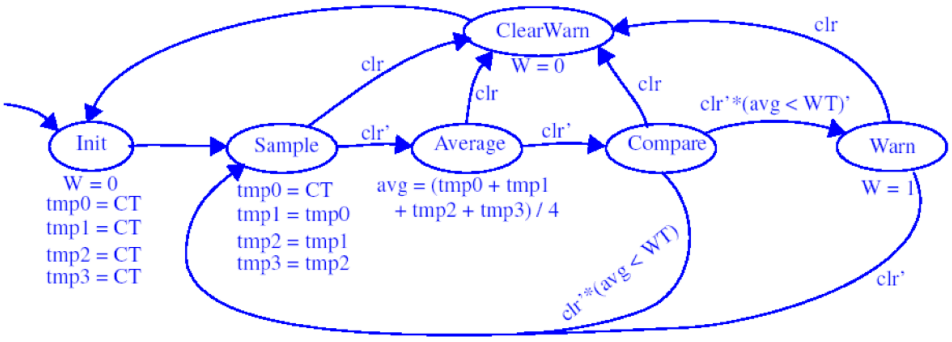
High-Level State Machine



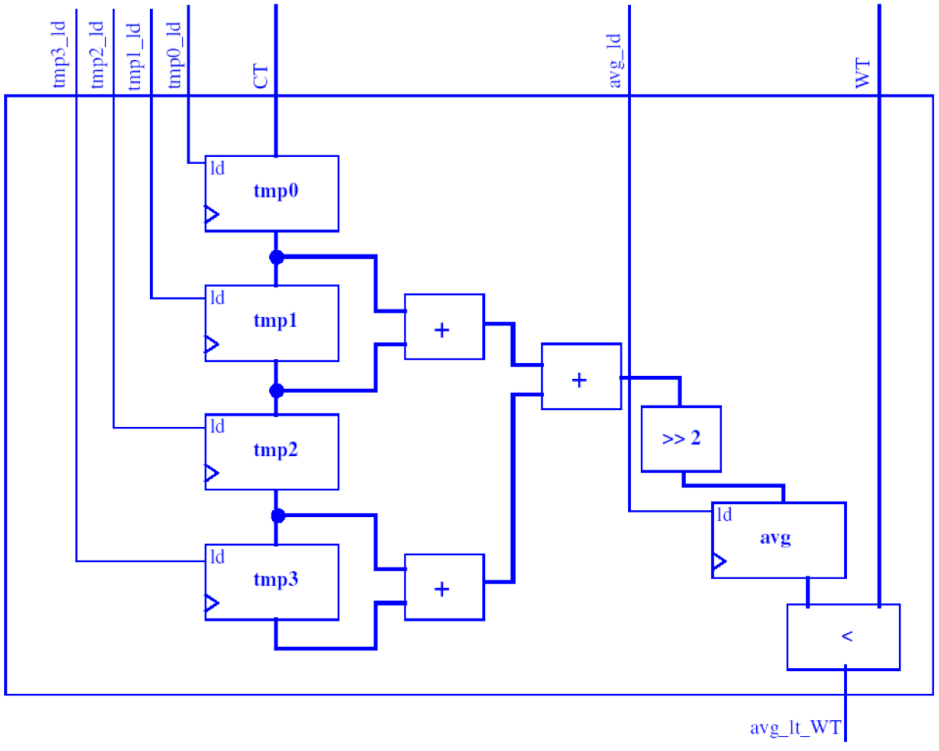
5.17)

Step 1 - Capture a high-level state machine

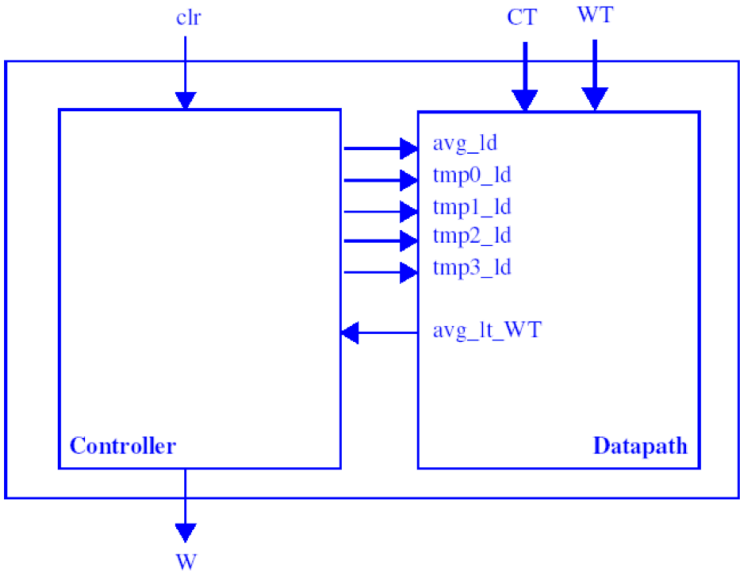
Inputs: CT, WT (32 bits); clr (bit)
Outputs: W (bit)
Local Registers: tmp0, tmp1, tmp2, tmp3, avg (32 bits)



Step 2 - Create a datapath

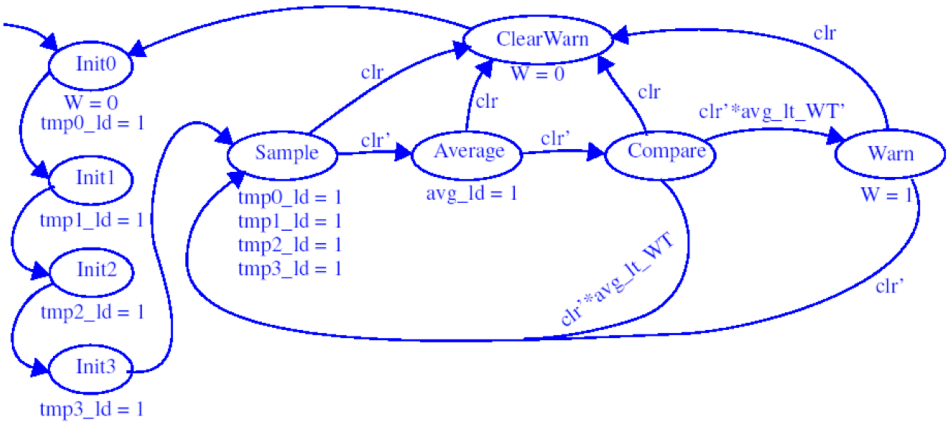


Step 3 - Connect the datapath to a controller



Step 4 - Derive the controller's FSM

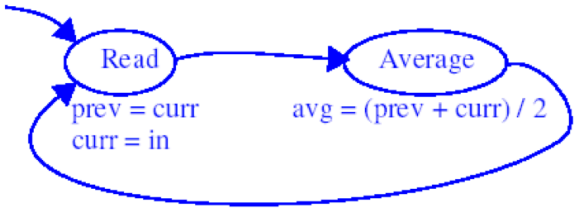
Inputs: CT, WT (32 bits); clr (bit)
Outputs: W (bit)
Local Registers: tmp0, tmp1, tmp2, tmp3, avg (32 bits)



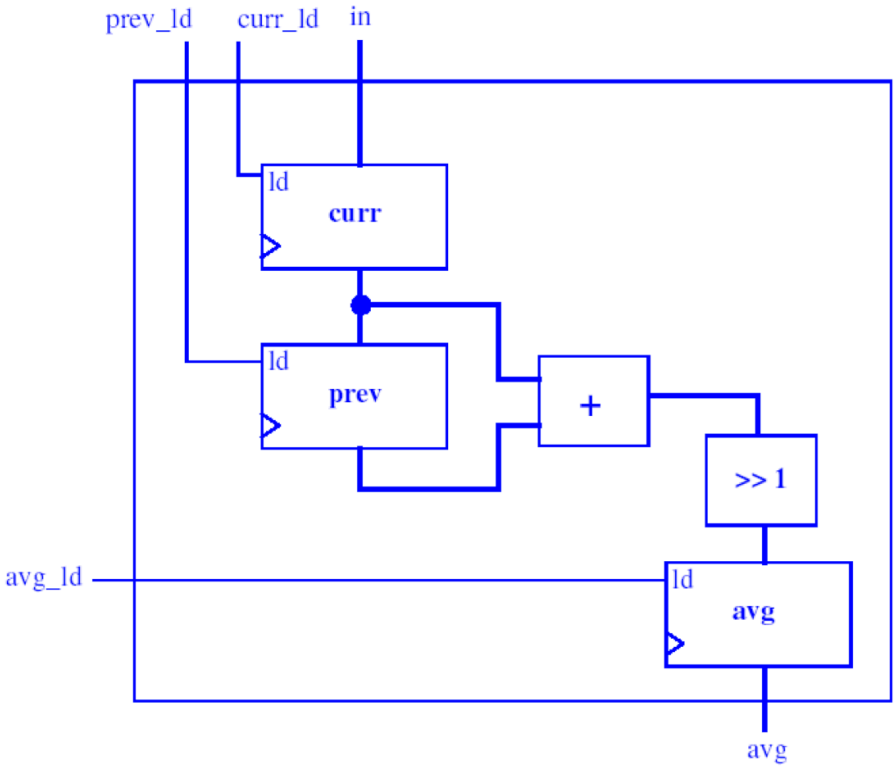
5.18)

Step 1 - Capture a high-level state machine

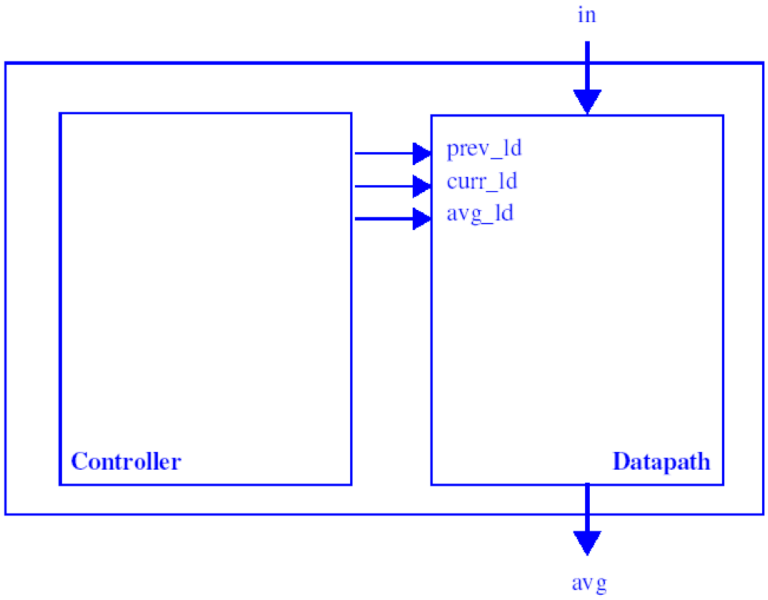
Inputs: in (32 bits)
Outputs: avg (32 bits)
Local Registers: curr, prev (32 bits)



Step 2 - Create a datapath

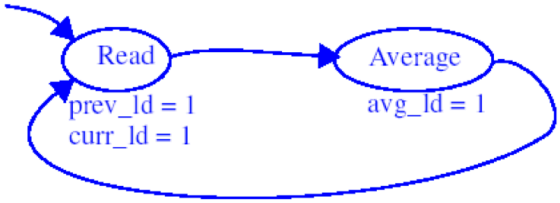


Step 3 - Connect the datapath to a controller



Step 4 - Derive the controller's FSM

Inputs: none
Outputs: curr_ld, prev_ld, avg_ld (bit)



8.9)

```
MOV R0, 0
MOV R1, 1
ADD R0, R0, R1
MOV 0, R0
```

8.10)

```
MOV R0, 1
ADD R0, R0, R0
MOV R1, 2
ADD R0, R0, R1
MOV 4, R0
```

8.12)

1) The contents of D[1] is loaded into RF[0]
2) The contents of D[9] is loaded into RF[1]
3) RF[0] is updated with the value RF[0] + RF[1]

8.13)

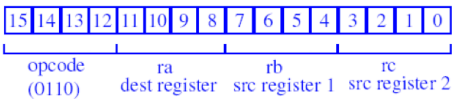
1. R6 gets loaded with the number 1
2. R5 gets loaded with the contents of D[9], 0
3. The PC gets loaded with PC + 2 - 1 (the offset of label1) since R5 is 0
4. R5 gets loaded with R5 + R6 (0 + 1 = 1)

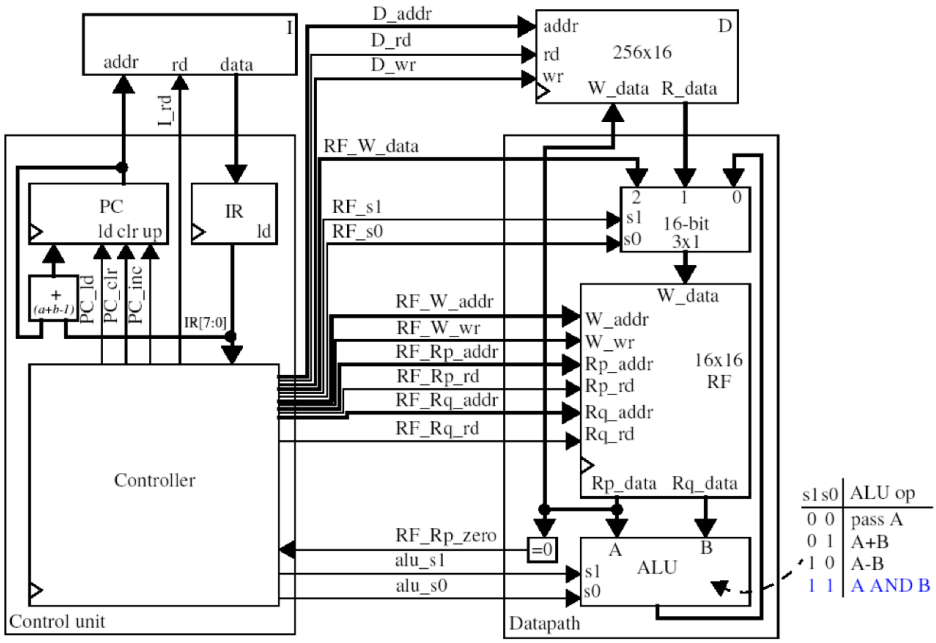
After the program completes, R5 is 1.

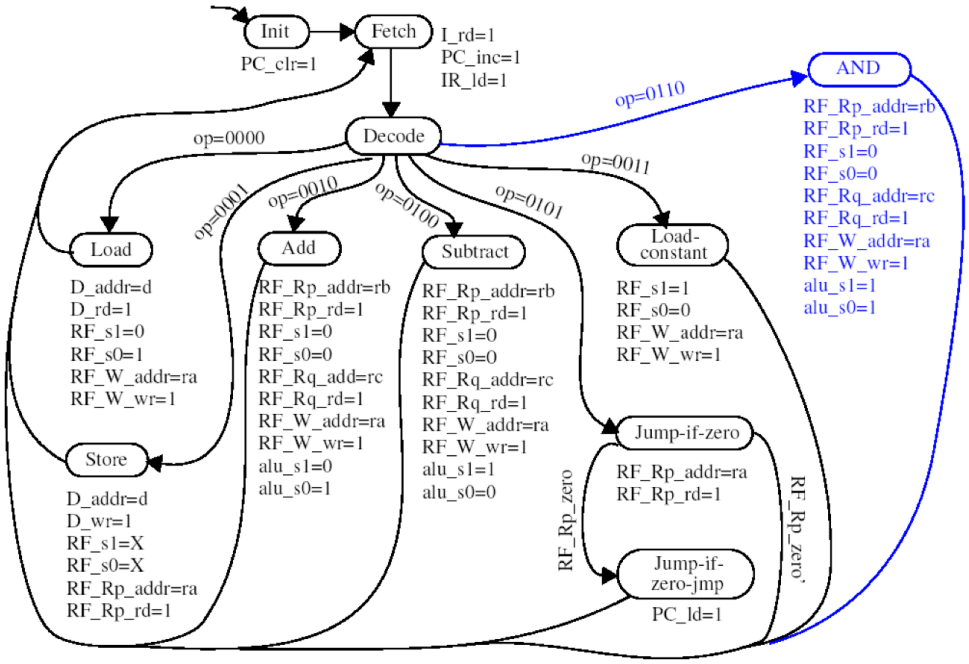
8.14)

We'll use the opcode 0110 for the AND operation. We'll modify the ALU to perform the AND operation when the ALU's s1s0=11.

AND ra, rb, rc



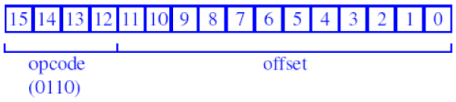


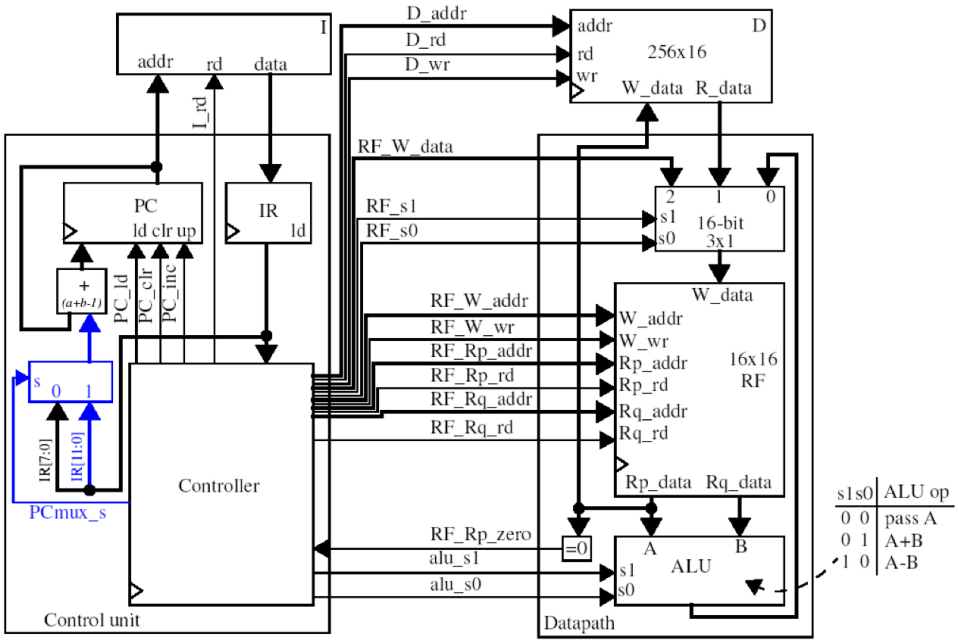


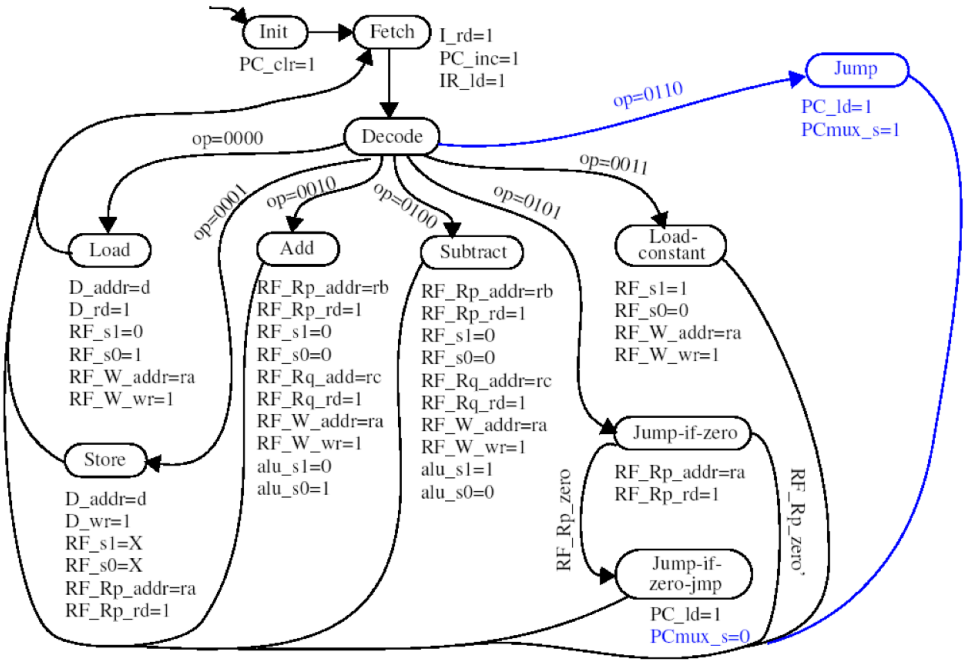
8.15)

We'll use opcode 0110.

JMP offset



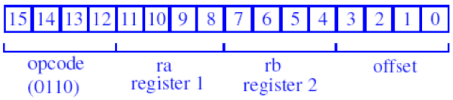


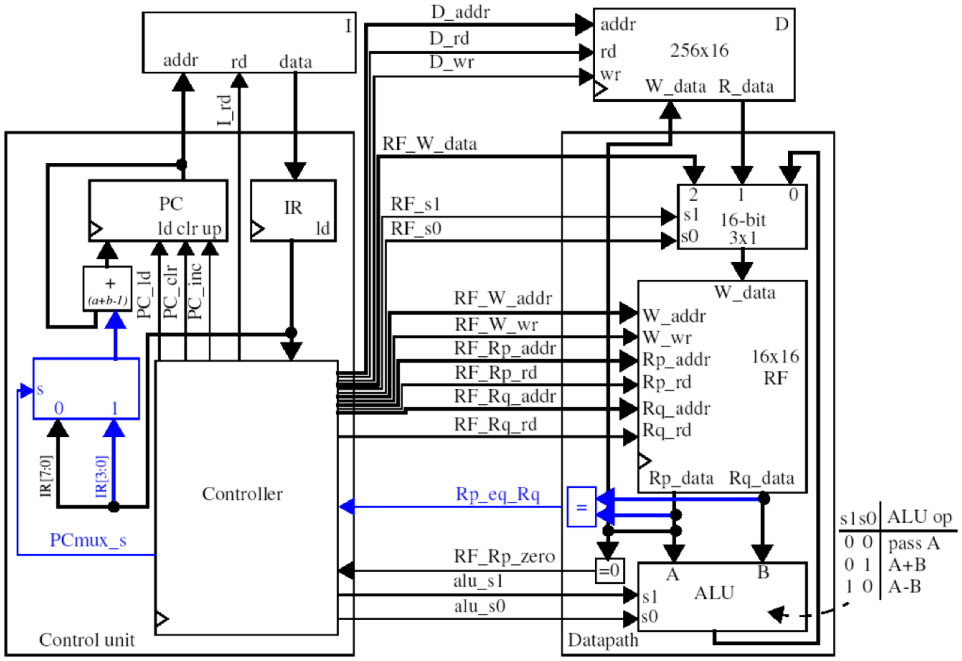


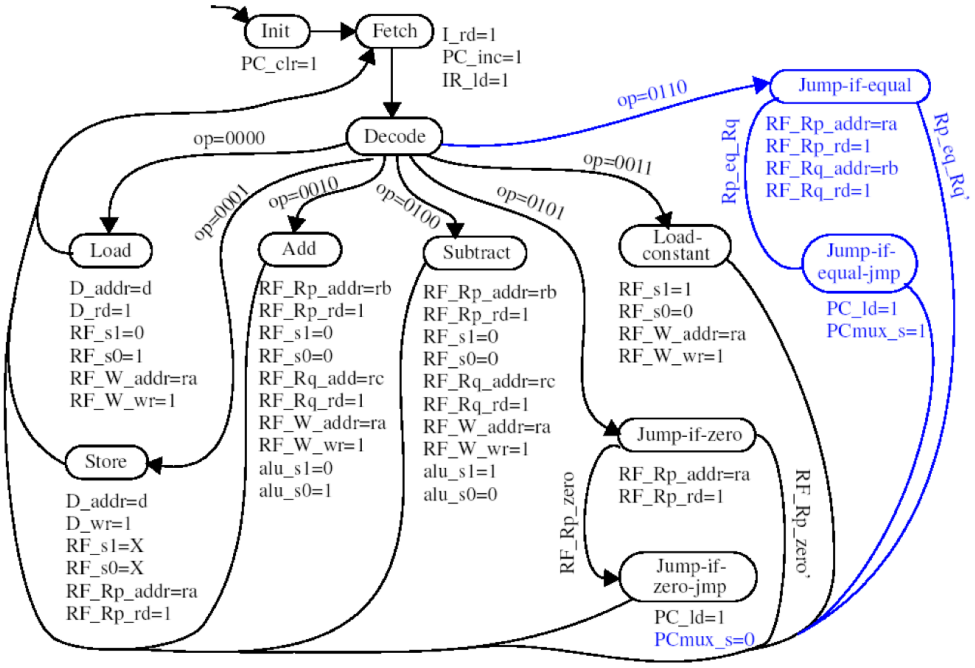
8.16)

We'll use opcode 0110.

JMPEQ ra, rb, offset







8.17)

```
        MOV R0, #0        // R0 is "i"
        MOV R1, #0        // R1 is "sum"
        MOV R2, #1        // R2 is the constant "1"
        MOV R3, 9         // R3 is "N" or "D[9]"
        MOV R4, #0        // R4 is the constant "0" (for looping)
loop:   SUB R5, R3, R0    // R4 = N - i
        JMPZ R5, done     // if i==N, end while loop
        ADD R1, R1, R0    // sum = sum + i
        ADD R0, R0, R2    // i = i + 1
        JMPZ R4, loop     // continue through while loop
done:
```

8.18)

```
        MOV R0, #0          // R0 is "i"
        MOV R1, #0          // R1 is "sum"
        MOV R2, #1          // R2 is the constant "1"
        MOV R3, 9           // R3 is "N" or "D[9]"
        MOV R4, #0          // R4 is the constant "0" (for looping)
loop:   JMPEQ R0, R3, done  // end while loop if i==N
        ADD R1, R1, R0      // sum = sum + i
        ADD R0, R0, R2      // i = i + 1
        JMPZ R4, loop       // continue through while loop
done:
```

8.22)

```
        MOV R0, #1          // R0 is the constant "1"
        MOV R1, 240         // R1 gets the value of I0
        MOV R2, 241         // R2 gets the value of I1
        AND R2, R1, R2      // R2 = I0 ANDI1
        MOV R1, 242         // R1 = I2
        AND R2, R1, R2      // R2 = R2 AND I2
        MOV R1, 243         // R1 = I3
        AND R2, R1, R2      // R2 = R2 AND I3
        MOV R1, 244         // R1 = I4
        AND R2, R1, R2      // R2 = R2 AND I4
        MOV R1, 245         // R1 = I5
        AND R2, R1, R2      // R2 = R2 AND I5
        MOV R1, 246         // R1 = I6
        AND R2, R1, R2      // R2 = R2 AND I6
        MOV R1, 247         // R1 = I6
        AND R2, R1, R2      // R2 = R2 AND I7
        SUB R2, R2, R0      // R2 = R2 - 1
        MOV R0, #0          // R0 is the constant "0"
        JMPZ R2, output     // If R2-1==0, then I7..I0 were all 1s
        JMPZ R0, done       // exit program
output: MOV 252, R0         // P4 = 0
done:
```

Questão 1

a) F (8 bits)

b) V

c) V

d) F (I não é afetada)

e) F (Z = 1)

f) F (STORE → escrever, FETCH → ler)

g) V

h) F (no jump, PC ≠ PC+1)

i) V

j) F (interrupções são assíncronas)

Questão 2

a) S8 = 05, S1 = 00 → S1 = S1 + S8 = 05, S8 = S8 − 01 = 04 → S1 = 05 + 04 = 09, S8 = 04 − 01 = 03

→ S1 = 09 + 03 = 12, S8 = 03 − 01 = 02 → S1 = 12 + 02 = 14, S8 = 02 − 01 = 01 → **S1 = 14 + 1 = 15, S8 = 01 − 01 = 00**

b) Mask = 1000 0000

S0 = 0000 0000

S0 = (1)000 0000 xor (0)000 0000

TOGGLE no MSB, independentemente do valor da entrada.

Questão 3

s1 = 1000 1000

s0 = 01 → sr0 s1 (shift right)

s1 = 0100 0100 c = 0

s0 = 02 → sl0 s1 (shift left)

s1 = 0001 0000 c = 1

s0 = 04 → rr s1 (rotate right)

s1 = 0100 0100 c = 0

s0 = 08 → rl s1 (rotate left)

s1 = 0001 0001 c = 1

Questão 4

Entrada = 01010011

Saída = 1100 1010

s1 = 0101 0011

s2 = 0000 0000

sr0 s1 → 0010 1001 c = 1

sla s2 → 0000 0001 c = 0

sr0 s1 → 0001 0100 c = 1

sla s2 → 0000 0011 c = 0

sr0 s1 → 0000 1010 c = 0

sla s2 → 0000 0110 c = 0

Questão 4 (cont.)

srθ s1 → 0000 0101 — c = 0
sla s2 → 0000 1100 — c = 0

srθ s1 → 0000 0010 — c = 1
sla s2 → 0001 1001 — c = 0

srθ s1 → 0000 0001 — c = 0
sla s2 → 0011 0010 — c = 0

srθ s1 → 0000 0000 — c = 1
sla s2 → 0110 0101 — c = 0

srθ s1 → 0000 0000 — c = 0
sla s2 → 1100 1010 — c = 0

(V)

(F) Em 2 ciclos.

(F) Circuito pode ir para um dentre vários estados possíveis.

(F) Pode ser sintetizado.

(F) 2500 instruções.

(F) Os conteúdos das chaves e dos leds serão armazenados nos endereços das portas de entrada e saída.

(V)

(V)

(F) Os dados de S0 são passados para a porta de saída.

(F) No jump, PC é diferente de PC + 1.

Questão 6:

```
1   switches DSIN $00
2   LEDS DSOUT $01
3
4   start:      IN s0,switches
5               LOAD s1, $f0  ; s1 = 1111 0000
6               COMP s0, 80   ; se s0 < 128, c = 1
7               CALL c, less ; se c = 1, vai para less
8               OUT s1, LEDS ; se s0 >= 128, c = 0 e a saida = 1111 0000
9               JUMP start   ; le nova entrada nas chaves
10  less:       LOAD s1, $0f ; se s0 < 128, s1 = 0000 1111
11              RET          ; retorna a rotina principal
```

OBS: será aceito também o código que testar o MSB.

Questão 7:

O trecho de código mostrado nesta questão está incorreto, mas seriam consideradas as respostas que contêm ao menos duas operações, independentemente da sequência mostrada no trecho. Segue abaixo o código corrigido:

```
; routine barrel shifter
; functions:
;           shift right (op: 00), shift left (op: 01)
;           rotate right (op: 10) and rotate left (op: 11)
; input registers:
;       s3 - data
;       s4 - amount (0 to 7)
;       s5 - operation
; output register:
;       s6 - result
;================================
; data RAM address alias
;================================
datain equ $00
amount equ $02
op equ $04
result equ $06
;================================
; register alias
;================================
data equ s0
i equ s2

sw1 equ s3
sw2 equ s4
sw3 equ s5
;================================
; port alias
;================================
sw1_port dsin $01
sw2_port dsin $02
sw3_port dsin $03
leds_port dsout $04
;================================
; main
;================================
```

```
; - clr_data_mem
; - read_switches
; - barrel_shifter
; - write_leds

              call clr_data_mem
forever:      call read_switches
              call barrel_shifter
              call write_leds
              jump forever
;================================
; clr_data_mem
;================================
clr_data_mem: load i,$10
              load data, $00
clr_mem_loop: store data,i
              sub i, $01
              jump nz,clr_mem_loop
              ret
;================================
; read_switches
;================================
read_switches: in sw1,sw1_port
               load data,sw1
               store data,datain
               in sw2,sw2_port
               load data,sw2
               store data,amount
               in sw3,sw3_port
               load data,sw3
               store data,op
               ret
;================================
; write_leds
;================================
write_leds: load s6, s3
            out s6, leds_port
            ret
;================================
; barrel_shifter
;================================
barrel_shifter:       load s6, $00
shift_right:          comp s5,$00 ; se s5 = 00
                      jump nz, shift_left
                      load i,s4   ; i = s4
barrel_right:         sr0 s3      ; desloca s3 uma vez para a direita
                      sub i,$01   ; decrementa i
                      jump nz,barrel_right ; continua deslocando ate
i=0
                      ret
shift_left:           comp s5,$01 ; se s5 = 01
                      jump nz, rotate_right
                      load i,s4
barrel_left:          sl0 s3      ; desloca s3 uma vez para a
esquerda
                      sub i,$01
                      jump nz,barrel_left
                      ret
rotate_right:         comp s5,$10 ; se s5 = 10
                      jump nz, rotate_left
                      load i,s4
barrel_rot_right:     rr s3
                      sub i,$01
```

```
                    jump nz, barrel_rot_right
                    ret
rotate_left:        comp s5,$11
                    ret nz
                    load i,s4
barrel_rot_left:    rl s3
                    sub i,$01
                    jump nz, barrel_rot_left
                    ret
```

Questão 8:

Código corrigido: rotina "fatorial" vai para a linha 14, e o programa está calculando o fatorial de n-1, sendo n o número obtido das chaves.

```
;Variáveis locais utilizadas
i EQU s2
j EQU s8
t EQU s9

;Variveis globais
sw_in EQU sf

;Definição da porta de entrada (8 chaves => 8 bits)
sw_port dsin $01

;Definição da porta de saída (8 leds => saída)
led_port dsout $05

;Ler as chaves
in sw_in,sw_port

; Calcular fatorial
load s6, 1
load t,1
fatorial:   comp sw_in,t
            jump c, done
            load s3, s6
            load s4, t
            call mult_soft
            add t, 1
            jump fatorial

mult_soft:  load s5,00              ;limpa s5
            load i,08               ;inicializa o loop

mult_loop:  sr0 s4                  ;desloca LSB para o carry
            jump nc, shift_prod     ;lsb é 0
            add s5,s3               ;lsb é 1

shift_prod: sra s5      ;desloca o byte superior para a direita
                        ;carry => MSB, LSB => carry
            sra s6      ;desloca o byte inferior para a direita
                        ;LSB de s5 => MSB de s6
            sub i,01                ;decrementa o loop
            jump nz,mult_loop       ;repetir até que i=0
            ret

done:       out s3,led_port
```

Exemplo: sw_in = 5

s6 = 1
t = 1
fatorial: sw_in < t? (5 <1?) Não
s3 = s6 = 1
s4 = t = 1
mult_soft: s5 = 1 * 1 = 1
ret

fatorial: t = t + 1 = 2
fatorial: sw_in < t? (5 <2?) Não
s3 = s6 = 1
s4 = t = 2
mult_soft: s6 = s3 * s4 = 1 * 2 = 2
ret

fatorial: t = t + 1 = 3
fatorial: sw_in < t? (5 <3?) Não
s3 = s6 = 2
s4 = t = 3
mult_soft: s6 = s3 * s4 = 2 * 3 = 6
ret

fatorial: t = t + 1 = 4
fatorial: sw_in < t? (5 <4?) Não
s3 = s6 = 6
s4 = t = 4
mult_soft: s6 = s3 * s4 = 6 * 4 = 24
ret

fatorial: t = t + 1 = 5
fatorial: sw_in < t? (5 <5?) Não
s3 = s6 = 24
s4 = t = 5
mult_soft: s6 = s3 * s4 = 24 * 5 = 120
ret

fatorial: t = t + 1 = 6
fatorial: sw_in < t? (5 <6?) Sim
c = 1 -> jump done

done: s3 -> leds = 24 = 4!

Para mostrar o fatorial de n, a rotina done poderia mostrar o conteúdo de s6 nos leds.