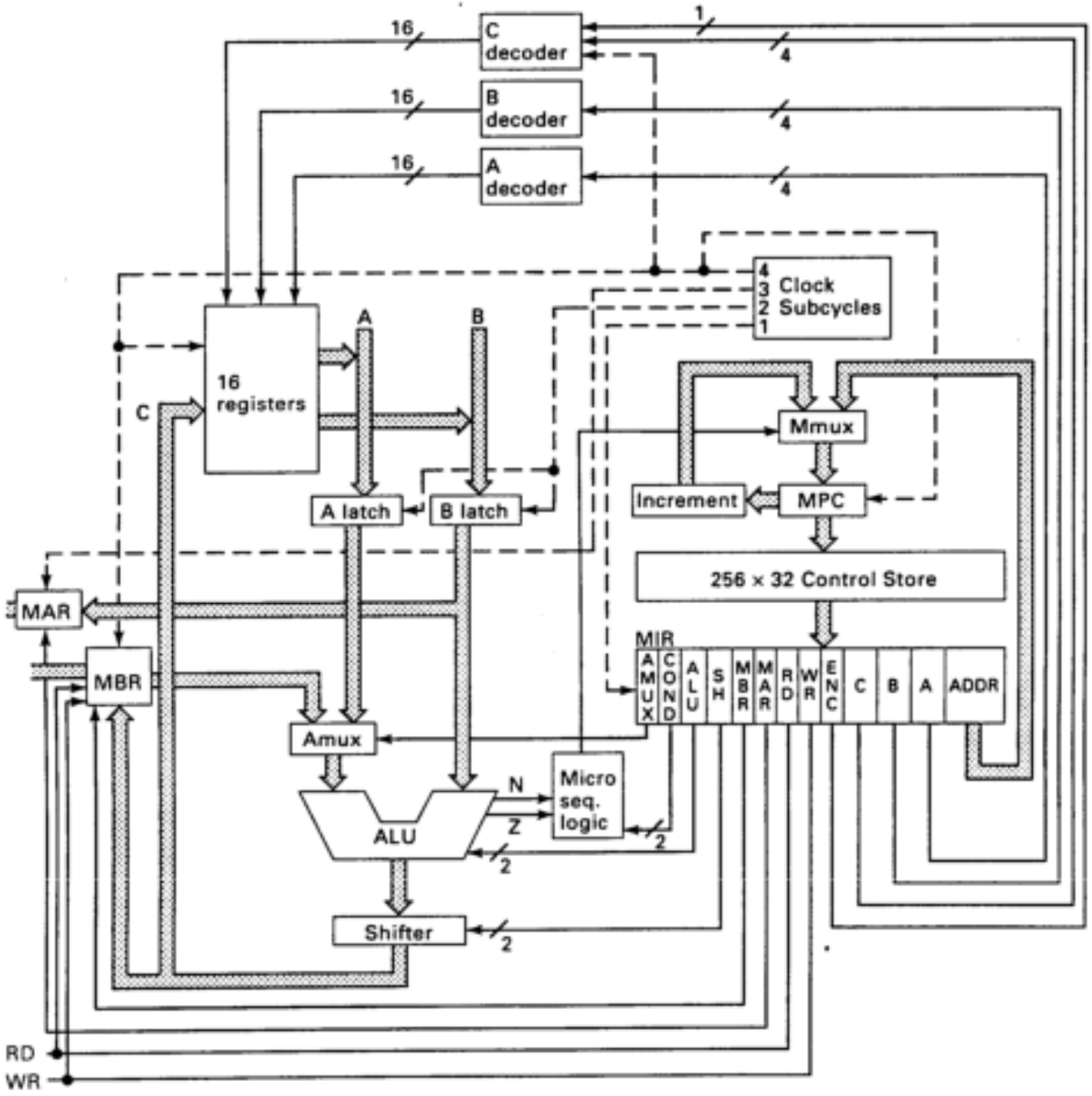
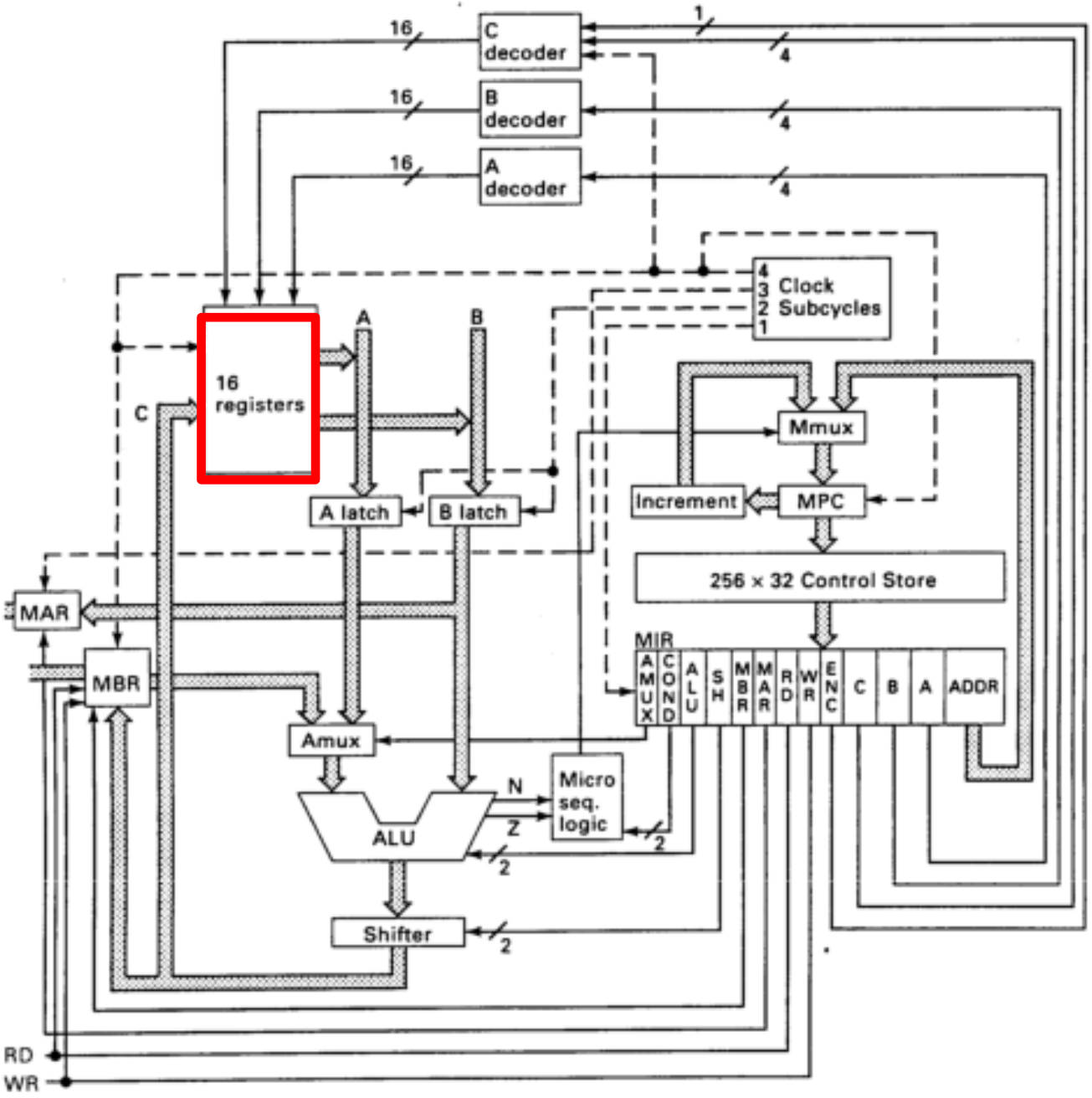
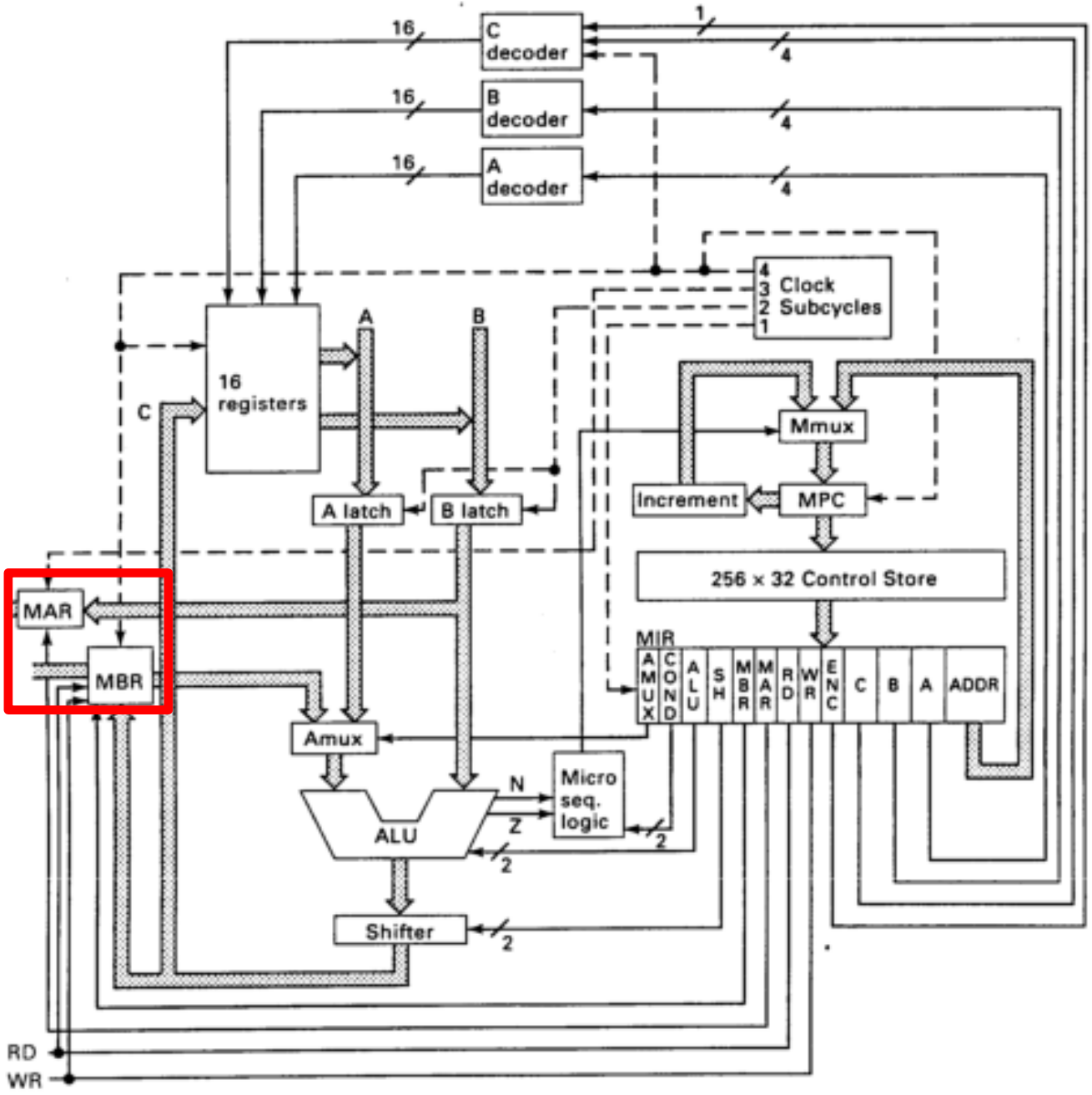
Teaching and Designing Microarchitecture in Racket

Jay McCarthy
UMass Lowell & PLT


16
C decoder
1
4
16
B decoder
4
16
A decoder
4
4
Clock
3
2
1
Subcycles
16
registers
C
A
B
Mmux
Increment
MPC
A latch
B latch
256 × 32 Control Store
MAR
MBR
MIR
AMUX
COND
ALU
SH
MBR
MAR
RD
WR
ENC
C
B
A
ADDR
Amux
N
Micro
seq.
logic
Z
ALU
2
2
Shifter
2
RD
WR


16
C decoder
1
4
16
B decoder
4
16
A decoder
4
4
Clock
3
2
1
Subcycles
16
registers
A
B
C
Mmux
Increment
MPC
A latch
B latch
256 × 32 Control Store
MAR
MBR
MIR
AMUX
COND
ALU
SH
MBR
MAR
RD
WR
ENC
C
B
A
ADDR
Amux
N
Micro
seq.
logic
Z
ALU
2
2
Shifter
2
RD
WR


C decoder
B decoder
A decoder
16
1
4
Clock
Subcycles
16 registers
A
B
C
A latch
B latch
Mmux
Increment
MPC
256 × 32 Control Store
MIR
AMUX
COND
ALU
SH
MBR
MAR
RD
WR
ENC
C
B
A
ADDR
MAR
MBR
Amux
ALU
N
Z
Micro seq. logic
Shifter
2
RD
WR

Binary	Mnemonic	Instruction	Meaning
0000xxxxxxxxxxxx	LODD	Load direct	$ac := m[x]$
0001xxxxxxxxxxxx	STOD	Store direct	$m[x] := ac$
0010xxxxxxxxxxxx	ADDD	Add direct	$ac := ac + m[x]$
0011xxxxxxxxxxxx	SUBD	Subtract direct	$ac := ac - m[x]$
0100xxxxxxxxxxxx	JPOS	Jump positive	**if** $ac \geq 0$ **then** $pc := x$
0101xxxxxxxxxxxx	JZER	Jump zero	**if** $ac = 0$ **then** $pc := x$
0110xxxxxxxxxxxx	JUMP	Jump	$pc := x$
0111xxxxxxxxxxxx	LOCO	Load constant	$ac := x\ (0 \leq x \leq 4095)$
1000xxxxxxxxxxxx	LODL	Load local	$ac := m[sp + x]$
1001xxxxxxxxxxxx	STOL	Store local	$m[x + sp] := ac$
1010xxxxxxxxxxxx	ADDL	Add local	$ac := ac + m[sp + x]$
1011xxxxxxxxxxxx	SUBL	Subtract local	$ac := ac - m[sp + x]$
1100xxxxxxxxxxxx	JNEG	Jump negative	**if** $ac < 0$ **then** $pc := x$
1101xxxxxxxxxxxx	JNZE	Jump nonzero	**if** $ac \neq 0$ **then** $pc := x$
1110xxxxxxxxxxxx	CALL	Call procedure	$sp := sp - 1;\ m[sp] := pc;\ pc := x$
1111000000000000	PSHI	Push indirect	$sp := sp - 1;\ m[sp] := m[ac]$
1111001000000000	POPI	Pop indirect	$m[ac] := m[sp];\ sp := sp + 1$
1111010000000000	PUSH	Push onto stack	$sp := sp - 1;\ m[sp] := ac$
1111011000000000	POP	Pop from stack	$ac := m[sp];\ sp := sp + 1$
1111100000000000	RETN	Return	$pc := m[sp];\ sp := sp + 1$
1111101000000000	SWAP	Swap ac, sp	$tmp := ac;\ ac := sp;\ sp := tmp$
11111100yyyyyyyy	INSP	Increment sp	$sp := sp + y\ (0 \leq y \leq 255)$
11111110yyyyyyyy	DESP	Decrement sp	$sp := sp - y\ (0 \leq y \leq 255)$

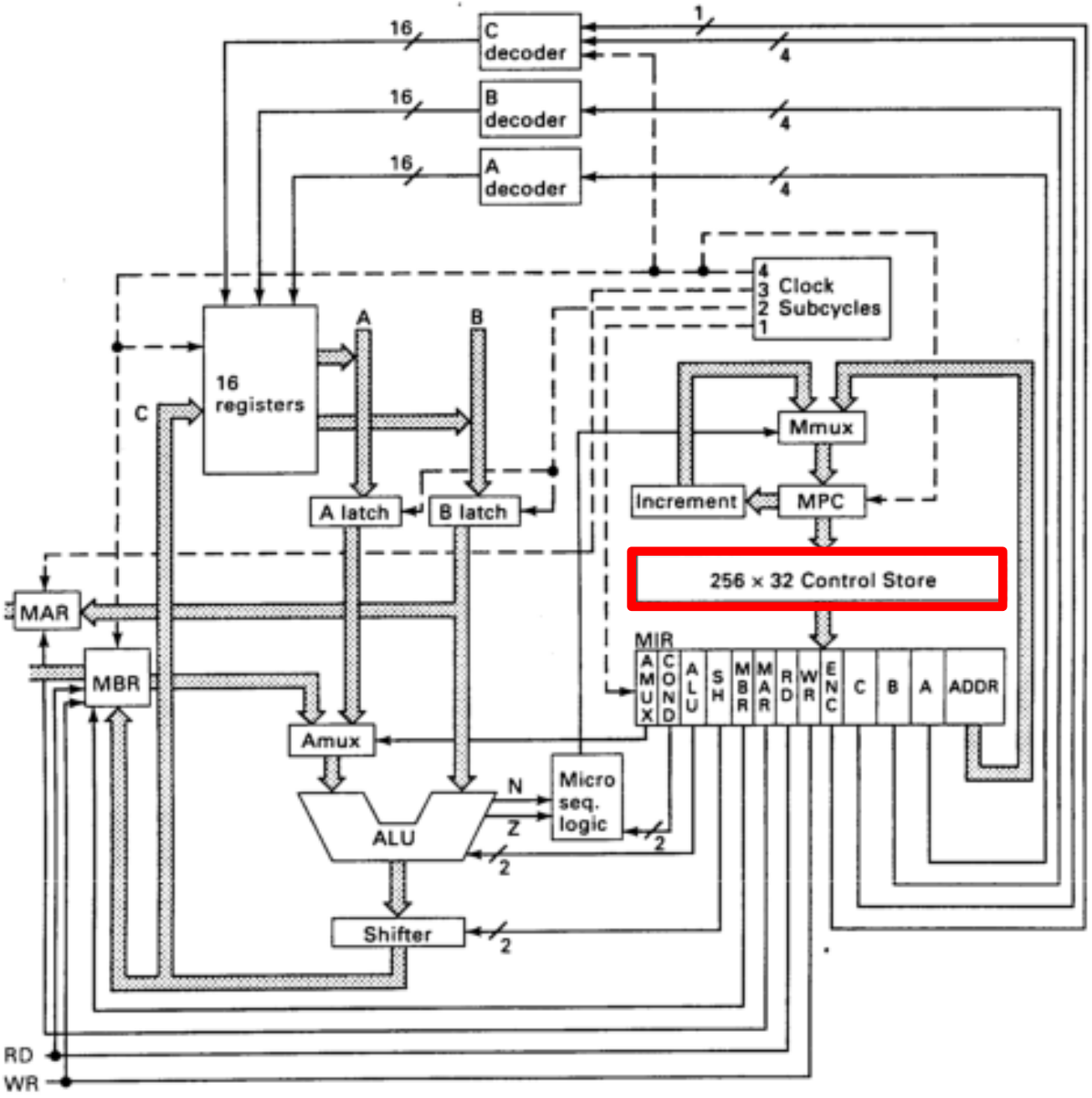
xxxxxxxxxxxx is a 12-bit machine address; in column 4 it is called x.
yyyyyyyy is an 8-bit constant; in column 4 it is called y.

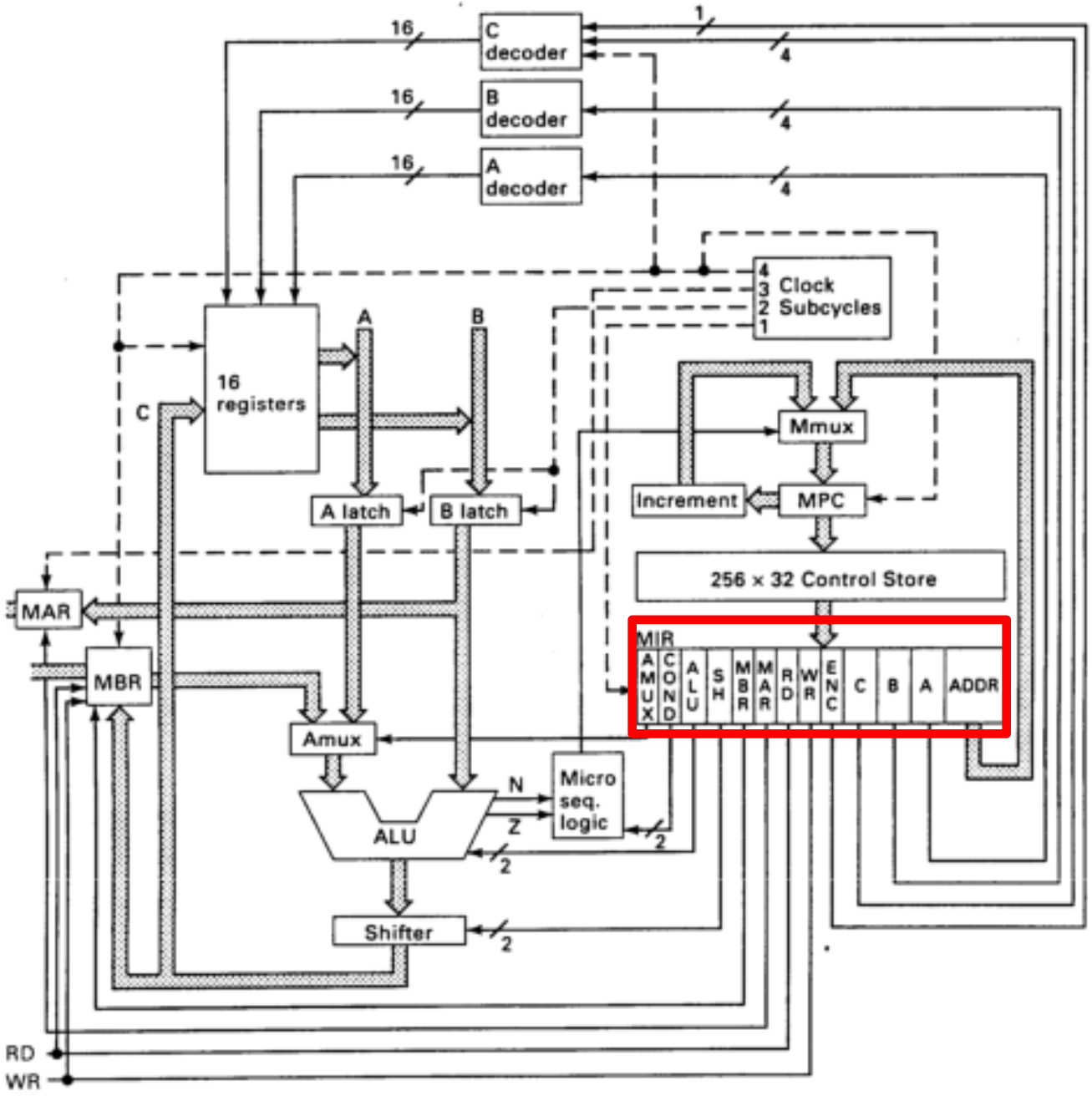
Binary	Mnemonic	Instruction	Meaning
0000xxxxxxxxxxxx	LODD	Load direct	$ac := m[x]$
0001xxxxxxxxxxxx	STOD	Store direct	$m[x] := ac$
0010xxxxxxxxxxxx	ADDD	Add direct	$ac := ac + m[x]$
0011xxxxxxxxxxxx	SUBD	Subtract direct	$ac := ac - m[x]$
0100xxxxxxxxxxxx	JPOS	Jump positive	**if** $ac \geq 0$ **then** $pc := x$
0101xxxxxxxxxxxx	JZER	Jump zero	**if** $ac = 0$ **then** $pc := x$
0110xxxxxxxxxxxx	JUMP	Jump	$pc := x$
0111xxxxxxxxxxxx	LOCO	Load constant	$ac := x\ (0 \leq x \leq 4095)$
1000xxxxxxxxxxxx	LODL	Load local	$ac := m[sp + x]$
1001xxxxxxxxxxxx	STOL	Store local	$m[x + sp] := ac$
1010xxxxxxxxxxxx	ADDL	Add local	$ac := ac + m[sp + x]$
1011xxxxxxxxxxxx	SUBL	Subtract local	$ac := ac - m[sp + x]$
1100xxxxxxxxxxxx	JNEG	Jump negative	**if** $ac < 0$ **then** $pc := x$
1101xxxxxxxxxxxx	JNZE	Jump nonzero	**if** $ac \neq 0$ **then** $pc := x$
1110xxxxxxxxxxxx	CALL	Call procedure	$sp := sp - 1;\ m[sp] := pc;\ pc := x$
1111000000000000	PSHI	Push indirect	$sp := sp - 1;\ m[sp] := m[ac]$
1111001000000000	POPI	Pop indirect	$m[ac] := m[sp];\ sp := sp + 1$
1111010000000000	PUSH	Push onto stack	$sp := sp - 1;\ m[sp] := ac$
1111011000000000	POP	Pop from stack	$ac := m[sp];\ sp := sp + 1$
1111100000000000	RETN	Return	$pc := m[sp];\ sp := sp + 1$
1111101000000000	SWAP	Swap ac, sp	$tmp := ac;\ ac := sp;\ sp := tmp$
11111100yyyyyyyy	INSP	Increment sp	$sp := sp + y\ (0 \leq y \leq 255)$
11111110yyyyyyyy	DESP	Decrement sp	$sp := sp - y\ (0 \leq y \leq 255)$

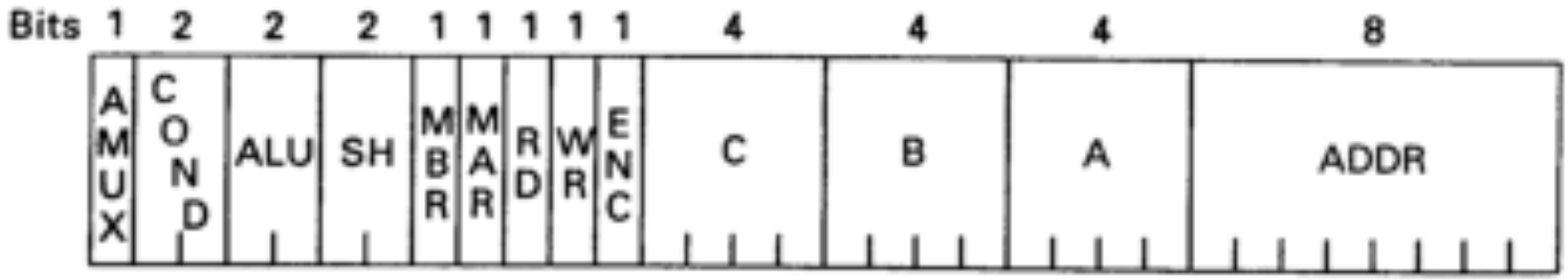
xxxxxxxxxxxx is a 12-bit machine address; in column 4 it is called x.
yyyyyyyy is an 8-bit constant; in column 4 it is called y.

```
       STOL 0      ; AC = 0 => M[SP] = Fib(0)
       LOCO 1      ; True => AC = 1
       STOL 1      ; AC = 1 => M[SP+1] = Fib(1)
loop:  ADDL 0      ; AC = Fib(n+1), M[SP] = Fib(n) => AC = Fib(n+2)
       STOL 2      ; AC = Fib(n+2) => M[SP+2] = Fib(n+2)
       INSP 1      ; increment n & SP
       JPOS loop:  ; If Fib(n+2) was negative, then n = 24
                                 =>
                          1001000000000000
                          0111000000000001
                          1001000000000001
                          1010000000000000
                          1001000000000010
                          1111110000000001
                          0100000000000011
```

```
(grammar
 (Program [() empty]
          [(Inst Program) (cons $1 $2)])
 (Arg [(NUM) $1] [(LABEL) (as-lref $1)])
 (Inst [(LABEL) (as-ldef $1)]
       [(     NUM) (as-lnum $1)]
       [(     STR) (as-lstr $1)]
       [( LOC NUM) (as-rloc $2)]
       [(LODD Arg) (as-iarg "0000" $2)]
       [(STOD Arg) (as-iarg "0001" $2)]
       [(ADDD Arg) (as-iarg "0010" $2)]
       [(SUBD Arg) (as-iarg "0011" $2)]
       [(JPOS Arg) (as-iarg "0100" $2)]
       [(JZER Arg) (as-iarg "0101" $2)]
       [(JUMP Arg) (as-iarg "0110" $2)]
       [(LOCO Arg) (as-iarg "0111" $2)]
       [(LODL Arg) (as-iarg "1000" $2)]
       [(STOL Arg) (as-iarg "1001" $2)]
       [(ADDL Arg) (as-iarg "1010" $2)]
       [(SUBL Arg) (as-iarg "1011" $2)]
       [(JNEG Arg) (as-iarg "1100" $2)]
       [(JNZE Arg) (as-iarg "1101" $2)]
       [(CALL Arg) (as-iarg "1110" $2)]
       [(PSHI) (as-inst "1111000000000000")]
       [(POPI) (as-inst "1111001000000000")]
       [(PUSH) (as-inst "1111010000000000")]
       [( POP) (as-inst "1111011000000000")]
       [(RETN) (as-inst "1111100000000000")]
       [(SWAP) (as-inst "1111101000000000")]
       [(INSP Arg) (as-iarg "11111100" $2)]
       [(DESP Arg) (as-iarg "11111110" $2)]
       [(HALT) (as-inst "1111111100000000")]))
```


16
C decoder
1
4
16
B decoder
4
16
A decoder
4
4
Clock
3
2
1
Subcycles
16
registers
C
A
B
Mmux
Increment
MPC
A latch
B latch
256 × 32 Control Store
MAR
MBR
MIR
AMUX
COND
ALU
SH
MBR
MAR
RD
WR
ENC
C
B
A
ADDR
Amux
N
Micro
seq.
logic
Z
ALU
2
2
Shifter
2
RD
WR


16
C decoder
1
4
16
B decoder
4
16
A decoder
4
4
Clock
3
2
1
Subcycles
16
registers
C
A
B
Mmux
Increment
MPC
A latch
B latch
256 × 32 Control Store
MAR
MBR
MIR
AMUX
COND
ALU
SH
MBR
MAR
RD
WR
ENC
C
B
A
ADDR
Amux
N
Micro
seq.
logic
Z
ALU
2
2
Shifter
2
RD
WR



AMUX

0 = A latch
1 = MBR

COND

0 = No jump
1 = Jump if N = 1
2 = Jump if Z = 1
3 = Jump always

ALU

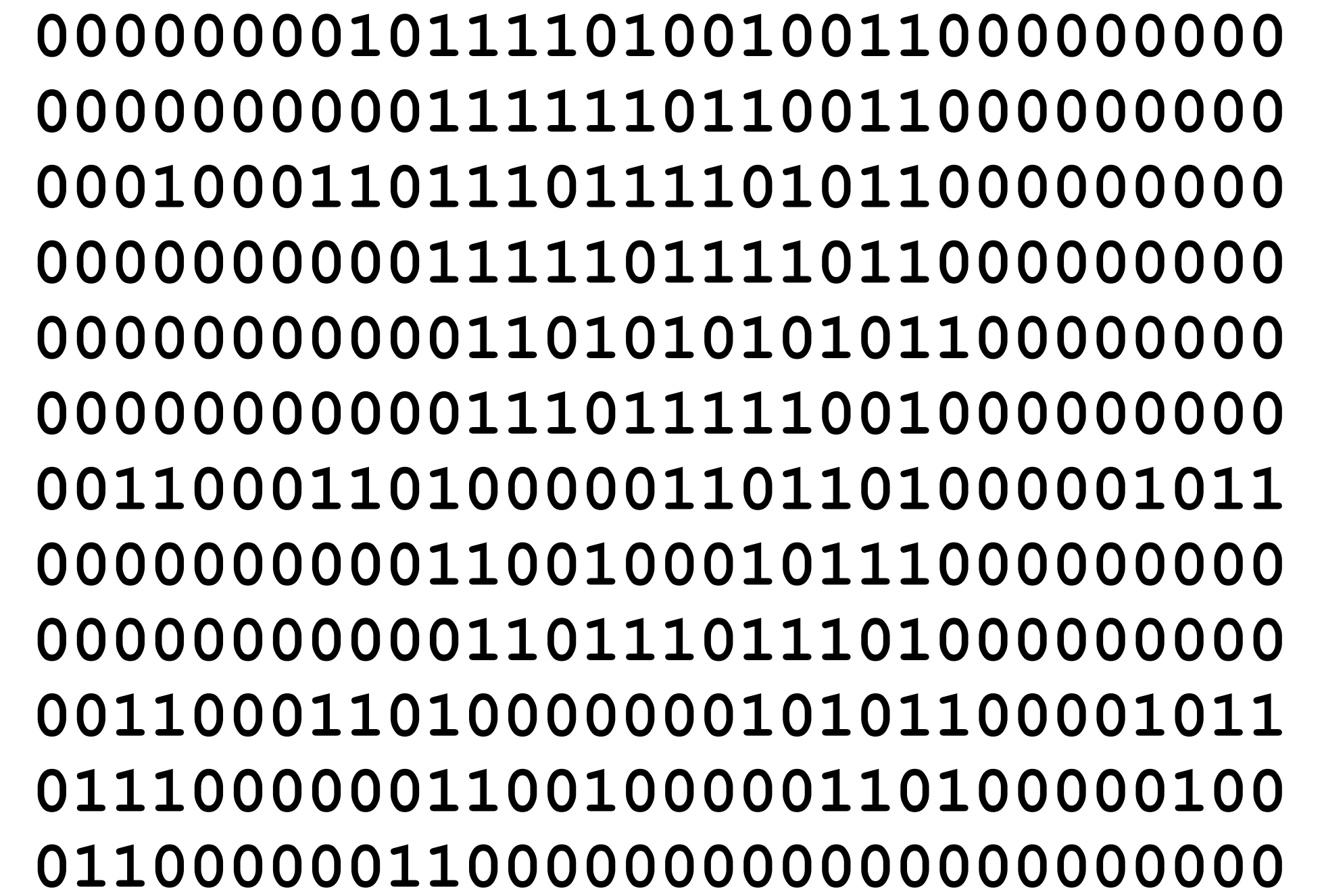
0 = A + B
1 = A AND B
2 = $\bar{A}$
3 = $\bar{A}$

SH

0 = No shift
1 = Shift right 1 bit
2 = Shift left 1 bit
3 = (not used)

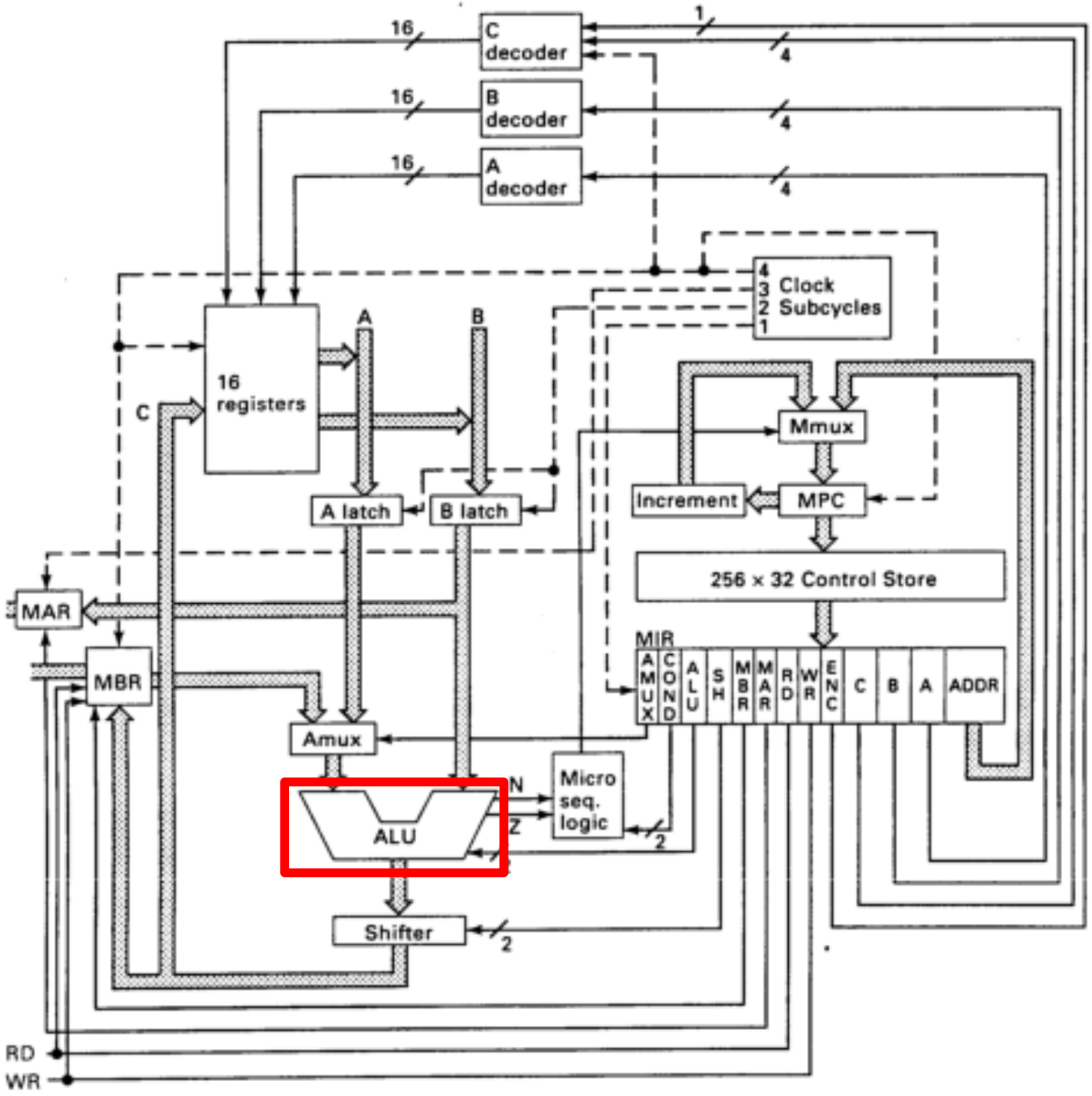
MBR, MAR, RD, WR,

0 = No
1 = Yes


0000000010111101001001100000000
0000000001111110110011000000000
0001000110111011110101100000000
0000000001111101111011000000000
0000000000011010101011000000000
0000000000011101111100100000000
0011000110100001101101000001011
0000000000110010001011100000000
0000000000011011101110100000000
0011000110100000010101100001011
0111000001100100001101000000100
0110000011000000000000000000000

```
START:
mar := sp; d := (1) + sp; wr;
f := (1) + (1); wr;
mbr := (1); b := (1); mar := d; wr;
e := f + (1); wr;
LOOP:
a := a + b;
d := f + sp;
mar :=  d; mbr := a; wr; if n then goto DONE;
sp := sp + e; wr;
b := b + a;
mar := sp; mbr := b; wr; if n then goto DONE;
sp := d; goto LOOP; wr;
DONE:
wr; rd; goto START;
```

```
(define (µcompile-reg L reg which)
  (hash-set1 L reg
             (match which
               ["pc" 'PC] ["ac" 'AC] ["sp" 'SP] ["ir" 'IR] ["tir" 'TIR]
               ["0" 'Z] ["1" 'P1] ["(-1)" 'N1] ["amask" 'AMASK] ["smask" 'SMASK]
               ["a" 'A] ["b" 'B] ["c" 'C] ["d" 'D] ["e" 'E] ["f" 'F"])))
(define (µcompile-b-bus L b-bus)
  (µcompile-reg L 'B b-bus))
(define (µcompile-c-bus L c-bus)
  (µcompile-reg L 'C c-bus))
(define (µcompile-a-bus L a-bus)
  (match a-bus
    ['mbr
     (hash-set1 L 'AMUX 'MBR)]
    [_
     (µcompile-reg L 'A a-bus)]))
(define (µcompile-alu L alu)
  (match alu
    [(alu-plus a-bus b-bus)
     (hash-set1 (µcompile-b-bus (µcompile-a-bus L a-bus) b-bus) 'ALU '+)]
    [(alu-id a-bus)
     (hash-set1 (µcompile-a-bus L a-bus) 'ALU 'A)]
    [(alu-band a-bus b-bus)
     (hash-set1 (µcompile-b-bus (µcompile-a-bus L a-bus) b-bus) 'ALU '&)]
    [(alu-inv a-bus)
     (hash-set1 (µcompile-a-bus L a-bus) 'ALU '!)]))
(define (µcompile-sh L sh)
  (match-define (sh-op alu) sh)
  (hash-set1
   (µcompile-alu L alu) 'SH
   (match sh
     [(sh-id _) 'NS]
     [(sh-lshift _) 'LS]
     [(sh-rshift _) 'RS])))
(define (µcompile-comp label->idx L comp)
  (match comp
    [(mc-setc c-bus sh)
     (hash-set1 (µcompile-c-bus (µcompile-sh L sh) c-bus) 'ENC 'ENC)]
    [(mc-alu alu)
     (µcompile-alu L alu)]
    [(mc-mar b-bus)
     (hash-set1 (µcompile-b-bus L b-bus) 'MAR 'MAR)]
    [(mc-mbr sh)
     (hash-set1 (µcompile-sh L sh) 'MBR 'MBR)]
    [(mc-if cond label)
     (hash-set1
      (hash-set1 L 'COND cond)
      'ADDR
      (hash-ref label->idx label
                (λ ()
                  (error 'µcompile "L~a: Unknown label: ~v"
                         (µcompile-line) label))))]
    [(mc-rd) (hash-set1 L 'RD 'RD)]
    [(mc-wr) (hash-set1 L 'WR 'WR)]))
```


16
C decoder
1
4
16
B decoder
4
16
A decoder
4
4
Clock
3
2
1
Subcycles
16
registers
C
A
B
Mmux
Increment
MPC
A latch
B latch
256 × 32 Control Store
MAR
MBR
MIR
AMUX
COND
ALU
SH
MBR
MAR
RD
WR
ENC
C
B
A
ADDR
Amux
N
Micro
seq.
logic
Z
2
ALU
Shifter
2
RD
WR

```
; ALU : N N 2 -> N 1 1
(define (ALU A B Function-Select #;=> Out Negative? Zero?)
  (define N (length A))
  (Net ([TheSum N] [TheAnd N] [NotA N] [Function-Selects 4])
       (Adder/N A B FALSE #;=> GROUND TheSum)
       (And/N A B #;=> TheAnd)
       (Not/N A #;=> NotA)
       (Decoder/N Function-Select #;=> Function-Selects)
       (RegisterRead (list TheSum TheAnd A NotA) Function-Selects
                     #;=> Out)
       (IsZero? Out #;=> Zero?)
       (IsNegative? Out #;=> Negative?)))
```

```
(define-chk-num chk-alu
  #:N N
  #:in ([A N] [B N] [Function-Select 2])
  #:out ([Out N] Negative? Zero?)
  #:circuit ALU #:exhaust 5
  #:check
  (chk (vector Out Negative? Zero?)
       (let* ([Ans-premod
               (match Function-Select
                 [0 (+ A B)]
                 [1 (bitwise-and A B)]
                 [2 A]
                 [3 (bitwise-not A)])]
              [Ans
               (modulo Ans-premod (expt 2 N))])
         (vector Ans
                 (negative? (unsigned->signed N Ans))
                 (zero? Ans)))))
```

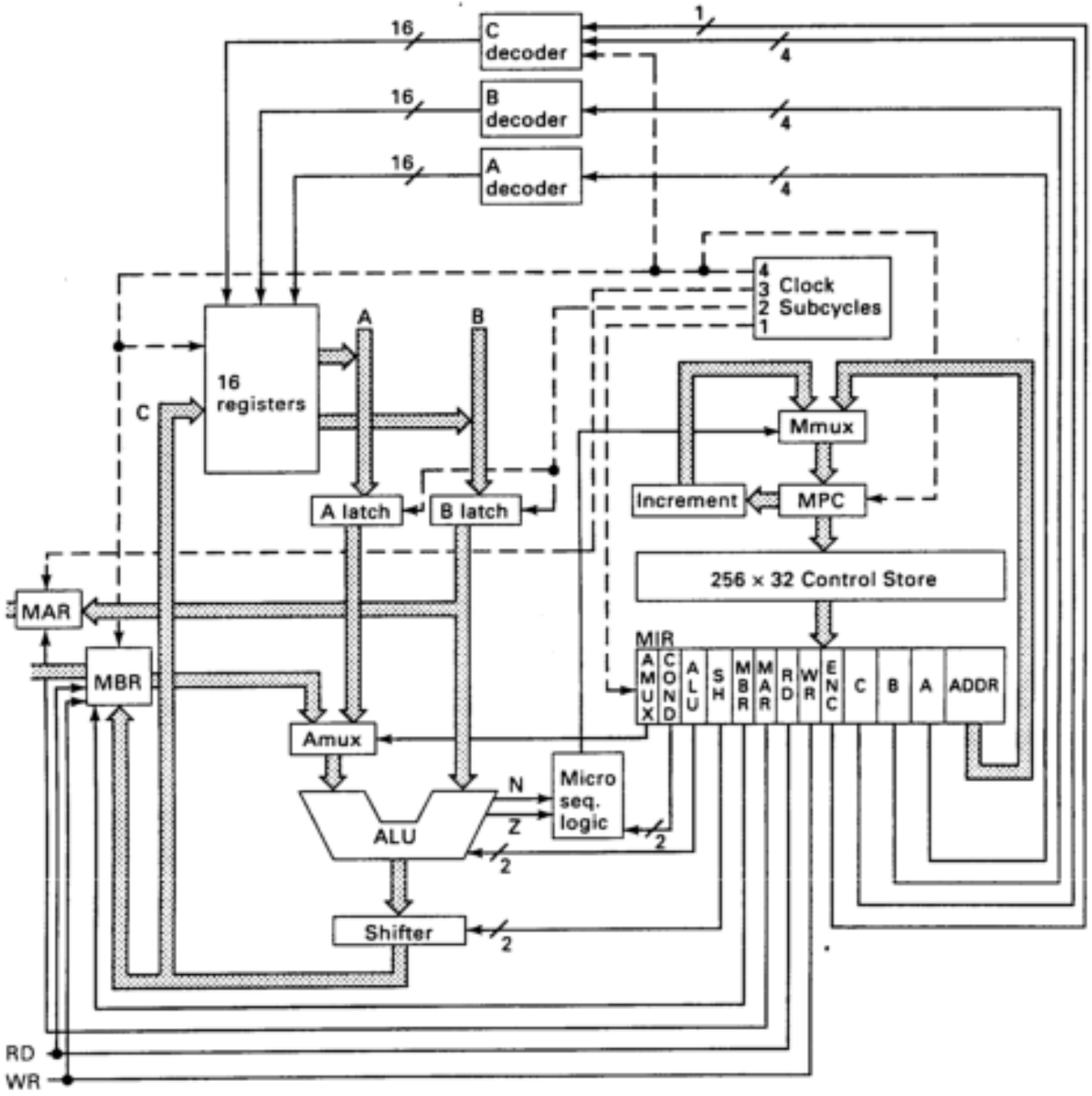
```
(define (MIC1 µCodeLength Microcode
              Registers MPC-out
              Read? Write?
              MAR MBR)
  (define µAddrSpace (ROM-AddrSpace Microcode))
  (define WordBits (length MBR))
  (define RegisterCount (length Registers))
  (define RegisterBits (integer-length (sub1 RegisterCount)))
  (define MIR:RD Read?)
  (define MIR:WR Write?)
  (define-wires
    Clock:1 Clock:2 Clock:3 Clock:4
    N Z MicroSeqLogic-out
    [pre-MIR µCodeLength] [MIR µCodeLength]
    MIR:AMUX [MIR:COND 2] [MIR:ALU 2] [MIR:SH 2]
    MIR:MBR MBR? MIR:MAR MAR?
    MIR:ENC
    [MIR:C RegisterBits] [MIR:B RegisterBits] [MIR:A RegisterBits]
    [MIR:ADDR µAddrSpace]
    [Mmux-out µAddrSpace]
    MPC-Inc-carry [MPC-Inc-out µAddrSpace]
    [Asel RegisterCount] [Bsel RegisterCount] [Csel RegisterCount]
    [A-Bus WordBits] [B-Bus WordBits] [C-Bus WordBits]
    [A-latch-out WordBits] [B-latch-out WordBits]
    [Amux-out WordBits] [ALU-out WordBits]
    Shifter-Left? Shifter-Right? Write-C?)
  (Net ()
       (Clock (list Clock:1 Clock:2 Clock:3 Clock:4))
       (ROM Microcode MPC-out pre-MIR)
       (Latch/N Clock:1 pre-MIR MIR)
       (Cut/N MIR
              (reverse
               (list MIR:AMUX MIR:COND MIR:ALU MIR:SH
                     MIR:MBR MIR:MAR MIR:RD MIR:WR
                     MIR:ENC MIR:C MIR:B MIR:A MIR:ADDR)))
       (Decoder/N MIR:A Asel)
       (Decoder/N MIR:B Bsel)
       (Decoder/N MIR:C Csel)
       (RegisterRead Registers Asel A-Bus)
       (RegisterRead Registers Bsel B-Bus)
       (Latch/N Clock:2 A-Bus A-latch-out)
       (Latch/N Clock:2 B-Bus B-latch-out)
       (And MIR:MAR Clock:3 MAR?)
       (Latch/N MAR? B-latch-out MAR)
       (Mux/N A-latch-out MBR MIR:AMUX Amux-out)
       (ALU Amux-out B-Bus MIR:ALU ALU-out N Z)
       (MicroSeqLogic N Z MIR:COND MicroSeqLogic-out)
       (Mux/N MPC-Inc-out MIR:ADDR MicroSeqLogic-out Mmux-out)
       (Decoder/N MIR:SH (list GROUND Shifter-Right? Shifter-Left? GROUND))
       (Shifter/N Shifter-Left? Shifter-Right? ALU-out C-Bus)
       (And MIR:MBR Clock:4 MBR?)
       (Latch/N MBR? C-Bus MBR)
       (And Clock:4 MIR:ENC Write-C?)
       (RegisterSet Write-C? C-Bus Csel Registers)
       (Latch/N Clock:4 Mmux-out MPC-out)
       (Increment/N MPC-out MPC-Inc-carry MPC-Inc-out)))
```

```
(define (simulate! net)
  (for ([n (in-list net)])
    (match-define (nand a b o) n)
    (box-set! o
              (not
               (and (unbox a)
                    (unbox b))))))
```

```
typedef uint64_t wire_t;
wire_t WIRES[111] = {0};
typedef uint8_t wireidx_t;
```

```
static void nand(wireidx_t aA, uint8_t aB,
                 wireidx_t bA, uint8_t bB,
                 wireidx_t oA, uint8_t oB) {
        uint8_t L = (WIRES[aA] & ((wire_t)1<<aB)) ? 1 : 0;
        uint8_t R = (WIRES[bA] & ((wire_t)1<<bB)) ? 1 : 0;
        uint8_t A = L & R ? 0 : 1;
        wire_t O = WIRES[oA];
        wire_t M = ((wire_t)1<<oB);
        WIRES[oA] = A ? (O | M) : (O & (~M));
}
```

```
void cycle() {
        nand(47, 49, 47, 49, 47, 50);
        nand(47, 49, 47, 49, 47, 51);
        nand(47, 51, 47, 51, 47, 52);
        nand(47, 53, 47, 53, 47, 54);
        nand(47, 53, 47, 53, 47, 55);
        nand(47, 55, 47, 55, 47, 56);
        nand(47, 50, 47, 54, 47, 57);
        nand(47, 57, 47, 57, 47, 58);
        nand(47, 50, 47, 56, 47, 59);
        nand(47, 59, 47, 59, 47, 60);
        nand(47, 52, 47, 54, 47, 61);
        nand(47, 61, 47, 61, 47, 62);
        nand(47, 52, 47, 56, 47, 63);
        nand(47, 63, 47, 63, 48, 0);
         ... 7000 more lines
       nand(0, 6, 47, 39, 47, 40);
       nand(47, 39, 47, 38, 47, 41);
       nand(47, 40, 47, 41, 24, 40);
       nand(0, 6, 47, 38, 47, 42);
       nand(47, 42, 47, 42, 47, 43);
       nand(0, 7, 47, 43, 47, 44);
       nand(0, 7, 47, 44, 47, 45);
       nand(47, 44, 47, 43, 47, 46);
       nand(47, 45, 47, 46, 24, 49);
       nand(0, 7, 47, 43, 47, 47);
       nand(47, 47, 47, 47, 47, 48); }
```


16
C decoder
1
4
16
B decoder
4
16
A decoder
4
4
Clock
3
2
1
Subcycles
16
registers
C
A
B
Mmux
Increment
MPC
A latch
B latch
256 × 32 Control Store
MAR
MBR
MIR
AMUX
COND
ALU
SH
MBR
MAR
RD
WR
ENC
C
B
A
ADDR
Amux
N
Micro
seq.
logic
Z
ALU
2
2
Shifter
2
RD
WR

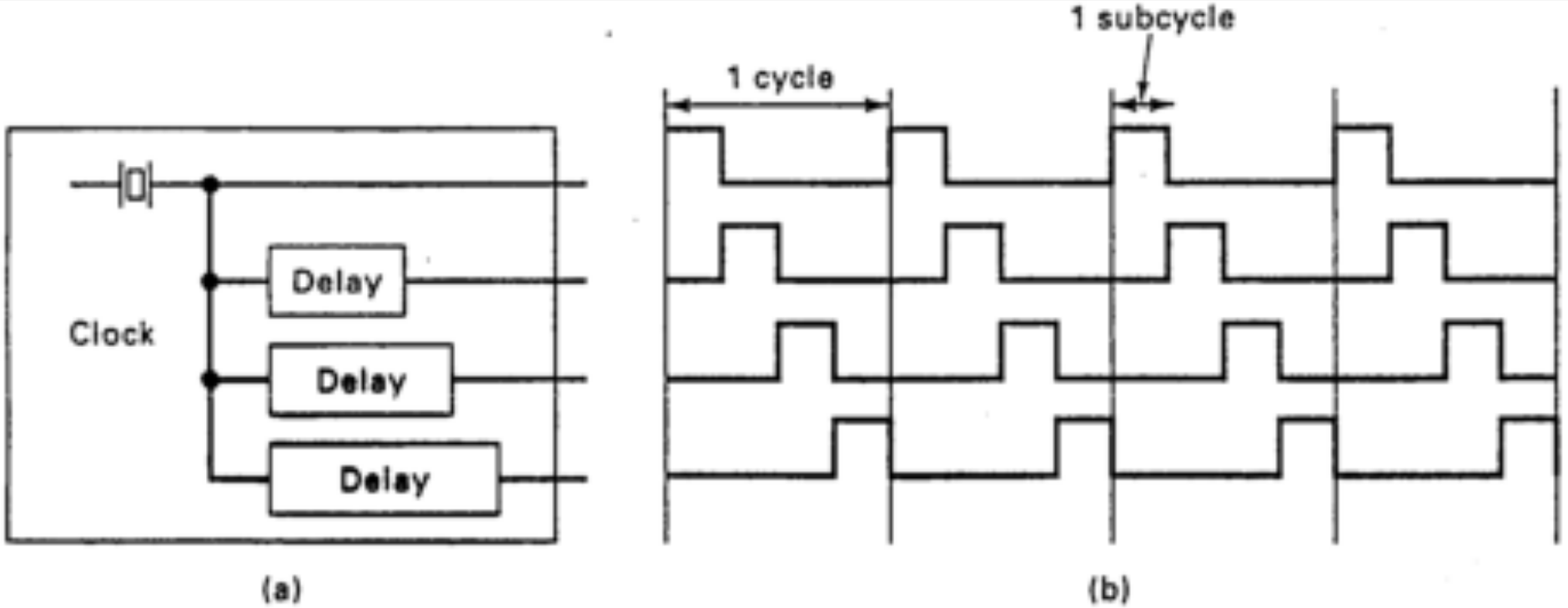
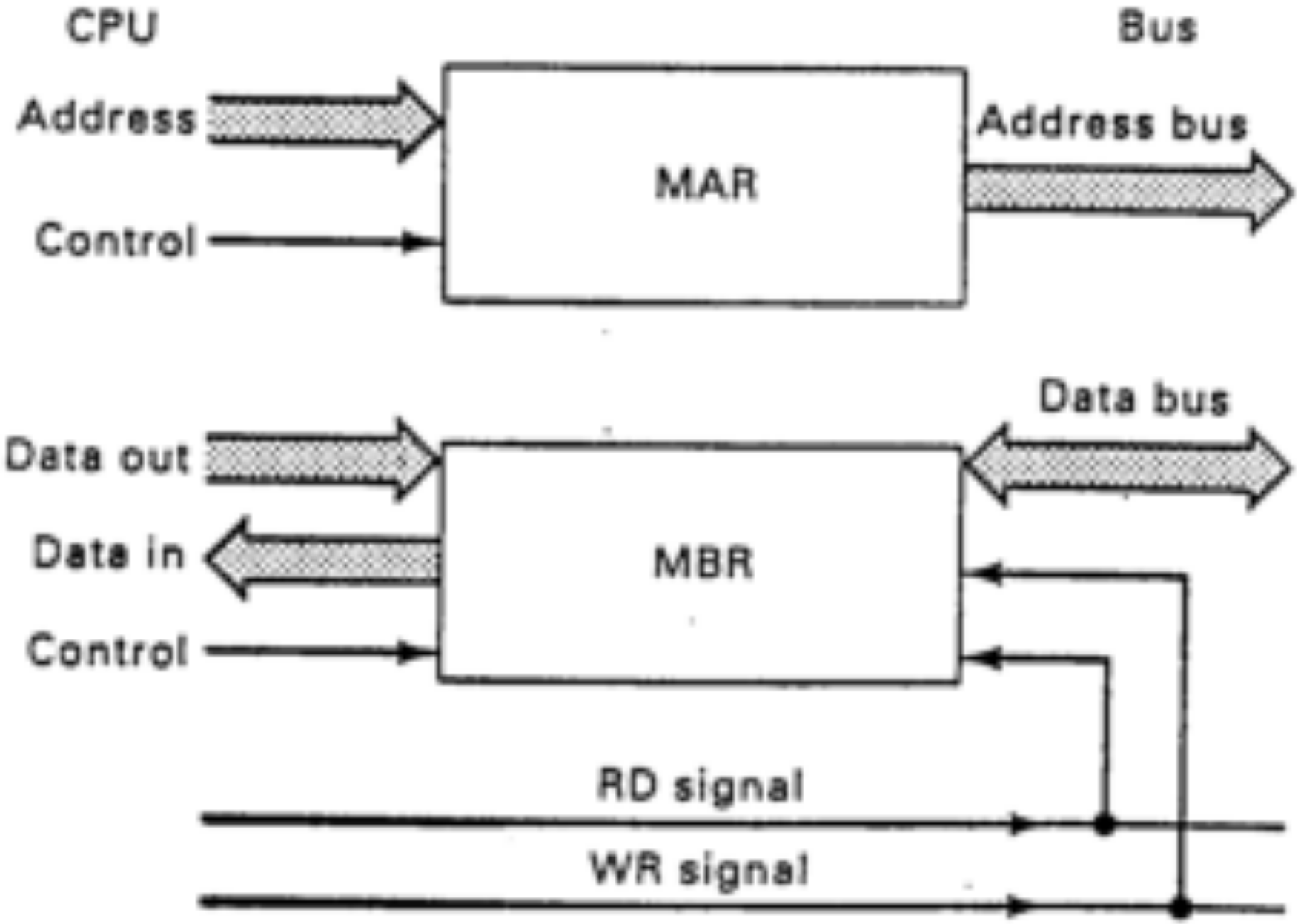
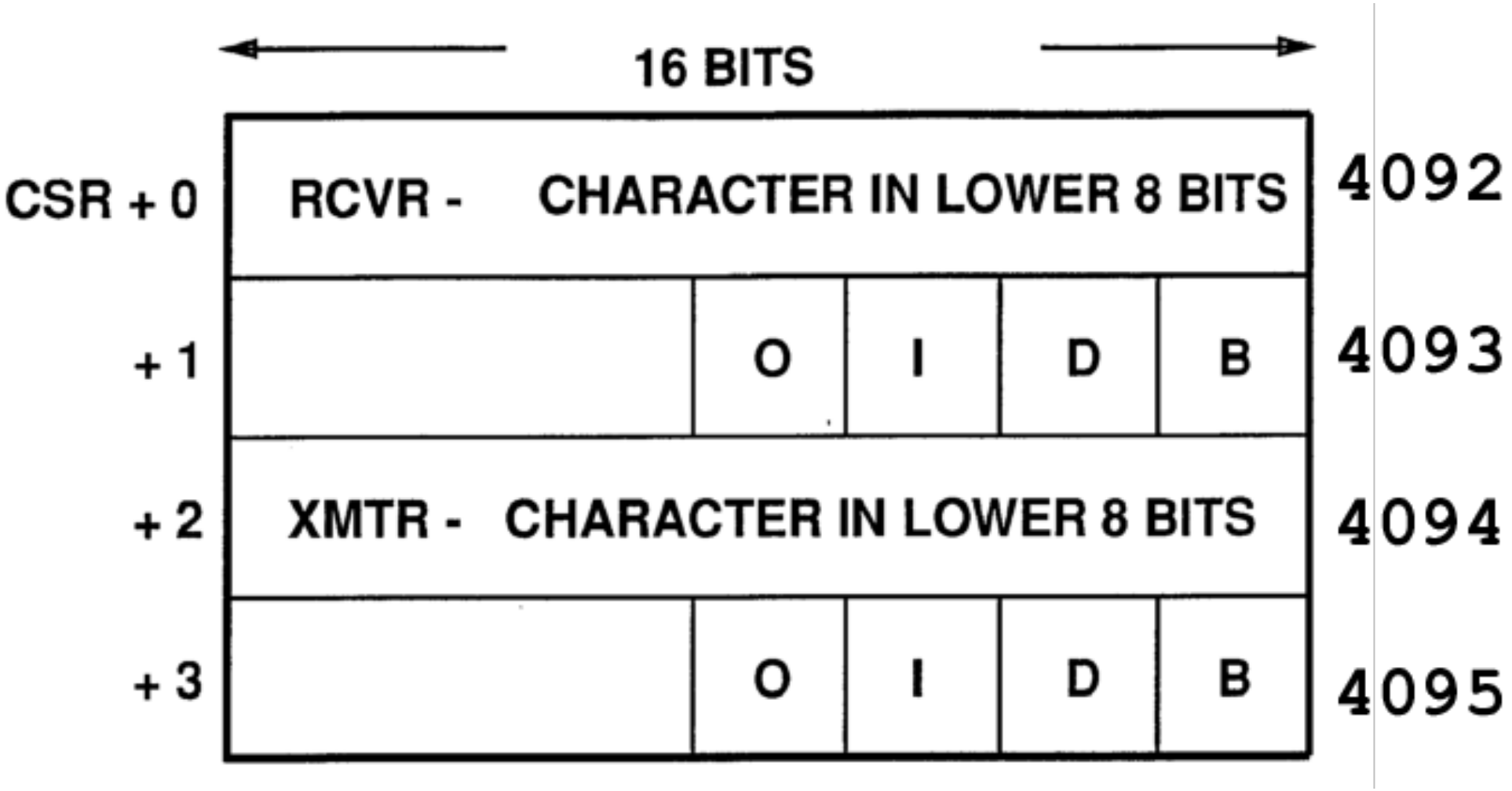


Fig. 4-5. (a) A clock with four outputs. (b) The output timing diagram.


CPU
Bus
Address
Address bus
MAR
Control
Data bus
Data out
Data in
MBR
Control
RD signal
WR signal


16 BITS
CSR + 0
RCVR - CHARACTER IN LOWER 8 BITS
4092
+ 1
O
I
D
B
4093
+ 2
XMTR - CHARACTER IN LOWER 8 BITS
4094
+ 3
O
I
D
B
4095