



CprE 281: Digital Logic

Instructor: Alexander Stoytchev

<http://www.ece.iastate.edu/~alexs/classes/>

Assembly Language

*CprE 281: Digital Logic
Iowa State University, Ames, IA
Copyright © Alexander Stoytchev*

Assembly Language

(for the i281 CPU)

CprE 281: Digital Logic
Iowa State University, Ames, IA
Copyright © Alexander Stoytchev

Intel 8086 Example

```
; _memcpy(dst, src, len)
; Copy a block of memory from one location to another.
;
; Entry stack parameters
;     [BP+6] = len, Number of bytes to copy
;     [BP+4] = src, Address of source data block
;     [BP+2] = dst, Address of target data block
;
; Return registers
;     AX = Zero
```

```
0000:1000                org     1000h           ; Start at 0000:1000h

0000:1000                _memcpy  proc
0000:1000 55              push    bp             ; Set up the call frame
0000:1001 89 E5            mov     bp,sp
0000:1003 06              push    es             ; Save ES
0000:1004 8B 4E 06        mov     cx,[bp+6]      ; Set CX = len
0000:1007 E3 11          jcxz   done           ; If len=0, return
0000:1009 8B 76 04        mov     si,[bp+4]      ; Set SI = src
0000:100C 8B 7E 02        mov     di,[bp+2]      ; Set DI = dst
0000:100F 1E            push   ds             ; Set ES = DS
0000:1010 07            pop     es

0000:1011 8A 04          loop   mov     al,[si]   ; Load AL from [src]
0000:1013 88 05          mov     [di],al       ; Store AL to [dst]
0000:1015 46            inc     si             ; Increment src
0000:1016 47            inc     di             ; Increment dst
0000:1017 49            dec     cx             ; Decrement len
0000:1018 75 F7          jnz    loop           ; Repeat the loop

0000:101A 07            done  pop     es             ; Restore ES
0000:101B 5D            pop     bp            ; Restore previous call frame
0000:101C 29 C0          sub     ax,ax         ; Set AX = 0
0000:101E C3            ret                    ; Return
0000:101F                end proc
```

Intel 8086 Example

Memory Address

```

; _memcpy(dst, src, len)
; Copy a block of memory from one location to another.
;
; Entry stack parameters
;     [BP+6] = len, Number of bytes to copy
;     [BP+4] = src, Address of source data block
;     [BP+2] = dst, Address of target data block
;
; Return registers
;     AX = Zero

0000:1000                org     1000h           ; Start at 0000:1000h

0000:1000                _memcpy  proc
0000:1000 55             push    bp           ; Set up the call frame
0000:1001 89 E5         mov     bp,sp
0000:1003 06             push    es           ; Save ES
0000:1004 8B 4E 06        mov     cx,[bp+6]    ; Set CX = len
0000:1007 E3 11          jcxz   done         ; If len=0, return
0000:1009 8B 76 04        mov     si,[bp+4]    ; Set SI = src
0000:100C 8B 7E 02        mov     di,[bp+2]    ; Set DI = dst
0000:100F 1E            push    ds           ; Set ES = DS
0000:1010 07            pop     es

0000:1011 8A 04          loop   mov     al,[si] ; Load AL from [src]
0000:1013 88 05          mov     [di],al     ; Store AL to [dst]
0000:1015 46            inc     si           ; Increment src
0000:1016 47            inc     di           ; Increment dst
0000:1017 49            dec     cx           ; Decrement len
0000:1018 75 F7          jnz    loop         ; Repeat the loop

0000:101A 07            done  pop     es           ; Restore ES
0000:101B 5D            pop     bp           ; Restore previous call frame
0000:101C 29 C0        sub     ax,ax       ; Set AX = 0
0000:101E C3            ret                ; Return
0000:101F                end proc

```

Intel 8086 Example

```
; _memcpy(dst, src, len)
; Copy a block of memory from one location to another.
;
; Entry stack parameters
;     [BP+6] = len, Number of bytes to copy
;     [BP+4] = src, Address of source data block
;     [BP+2] = dst, Address of target data block
;
; Return registers
;     AX = Zero
```

Machine Language

```
0000:1000                org     1000h        ; Start at 0000:1000h

0000:1000  _memcpy  proc
0000:1000  55      push    bp          ; Set up the call frame
0000:1001  89 E5   mov     bp,sp
0000:1003  06      push    es          ; Save ES
0000:1004  8B 4E 06 mov     cx,[bp+6]   ; Set CX = len
0000:1007  E3 11   jcxz   done        ; If len=0, return
0000:1009  8B 76 04 mov     si,[bp+4]   ; Set SI = src
0000:100C  8B 7E 02 mov     di,[bp+2]   ; Set DI = dst
0000:100F  1E      push    ds          ; Set ES = DS
0000:1010  07      pop     es

0000:1011  8A 04   loop   mov     al,[si] ; Load AL from [src]
0000:1013  88 05   loop   mov     [di],al  ; Store AL to [dst]
0000:1015  46     loop   inc     si      ; Increment src
0000:1016  47     loop   inc     di      ; Increment dst
0000:1017  49     loop   dec     cx      ; Decrement len
0000:1018  75 F7   loop   jnz    loop    ; Repeat the loop

0000:101A  07     done  pop     es          ; Restore ES
0000:101B  5D     done  pop     bp          ; Restore previous call frame
0000:101C  29 C0   done  sub     ax,ax       ; Set AX = 0
0000:101E  C3     done  ret
0000:101F                end proc
```

Intel 8086 Example

```
; _memcpy(dst, src, len)
; Copy a block of memory from one location to another.
;
; Entry stack parameters
;     [BP+6] = len, Number of bytes to copy
;     [BP+4] = src, Address of source data block
;     [BP+2] = dst, Address of target data block
;
; Return registers
;     AX = Zero
```

Assembly Language

```
0000:1000                org     1000h           ; Start at 0000:1000h

0000:1000                _memcpy proc
0000:1000 55                push   bp             ; Set up the call frame
0000:1001 89 E5             mov    bp,sp
0000:1003 06                push   es             ; Save ES
0000:1004 8B 4E 06         mov    cx,[bp+6]      ; Set CX = len
0000:1007 E3 11            jcxz  done            ; If len=0, return
0000:1009 8B 76 04         mov    si,[bp+4]      ; Set SI = src
0000:100C 8B 7E 02         mov    di,[bp+2]      ; Set DI = dst
0000:100F 1E                push   ds             ; Set ES = DS
0000:1010 07                pop    es

0000:1011 8A 04            loop   mov    al,[si]  ; Load AL from [src]
0000:1013 88 05            mov    [di],al        ; Store AL to [dst]
0000:1015 46                inc    si              ; Increment src
0000:1016 47                inc    di              ; Increment dst
0000:1017 49                dec    cx              ; Decrement len
0000:1018 75 F7            jnz   loop            ; Repeat the loop

0000:101A 07            done   pop    es        ; Restore ES
0000:101B 5D                pop    bp            ; Restore previous call frame
0000:101C 29 C0            sub    ax,ax         ; Set AX = 0
0000:101E C3                ret                  ; Return
0000:101F                end proc
```

Intel 8086 Example

```

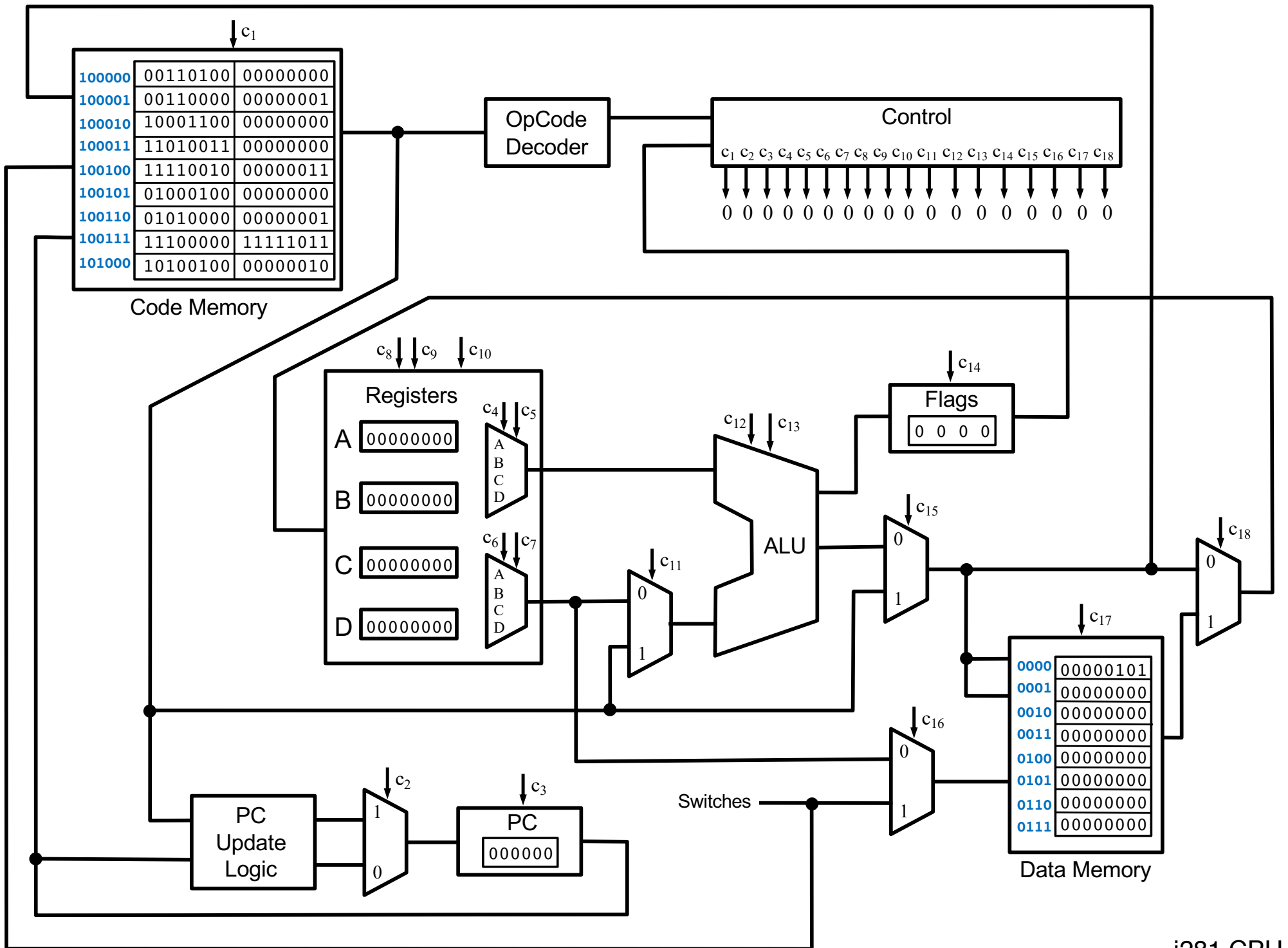
; _memcpy(dst, src, len)
; Copy a block of memory from one location to another.
;
; Entry stack parameters
;     [BP+6] = len, Number of bytes to copy
;     [BP+4] = src, Address of source data block
;     [BP+2] = dst, Address of target data block
;
; Return registers
;     AX = Zero

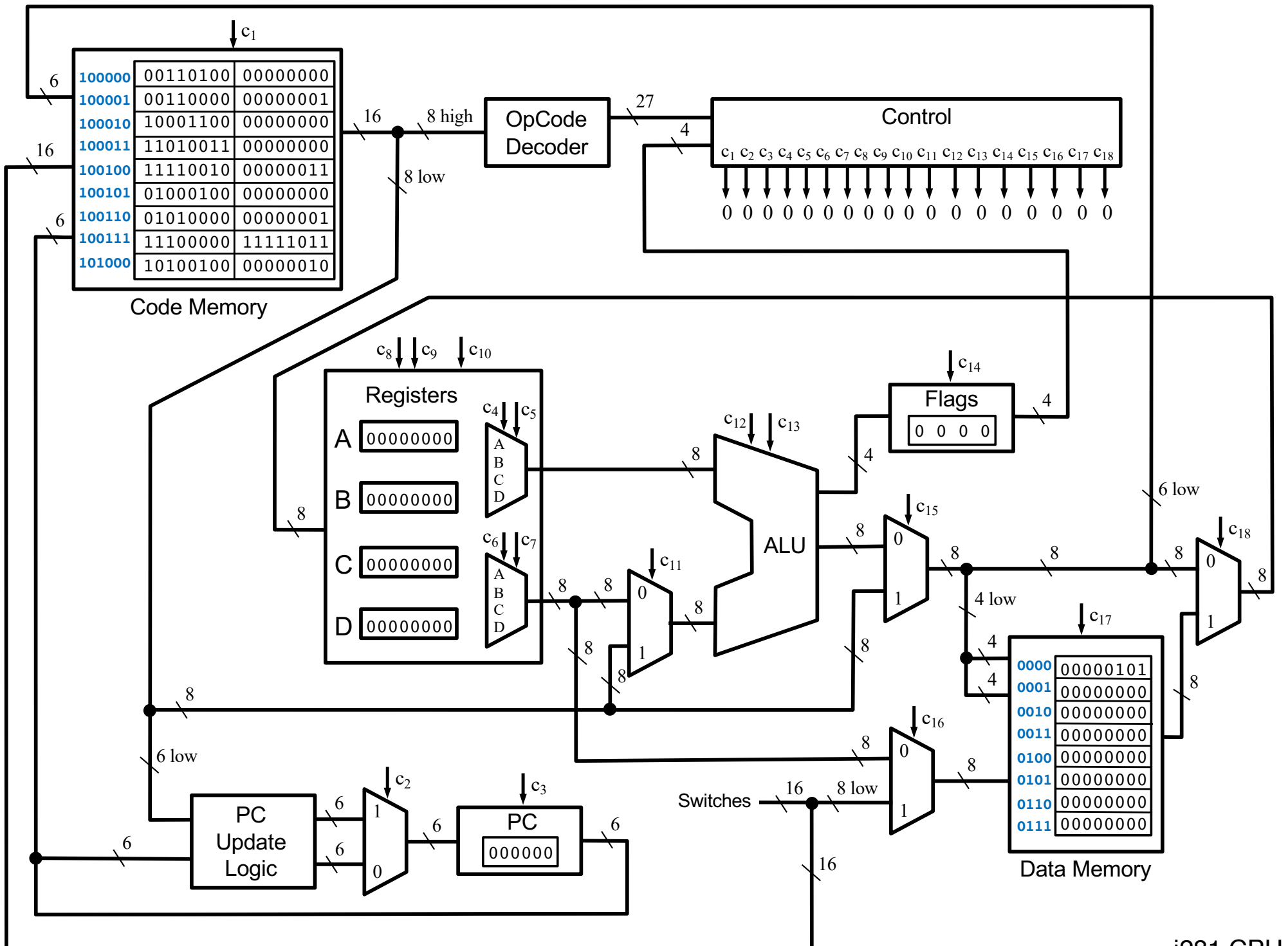
```

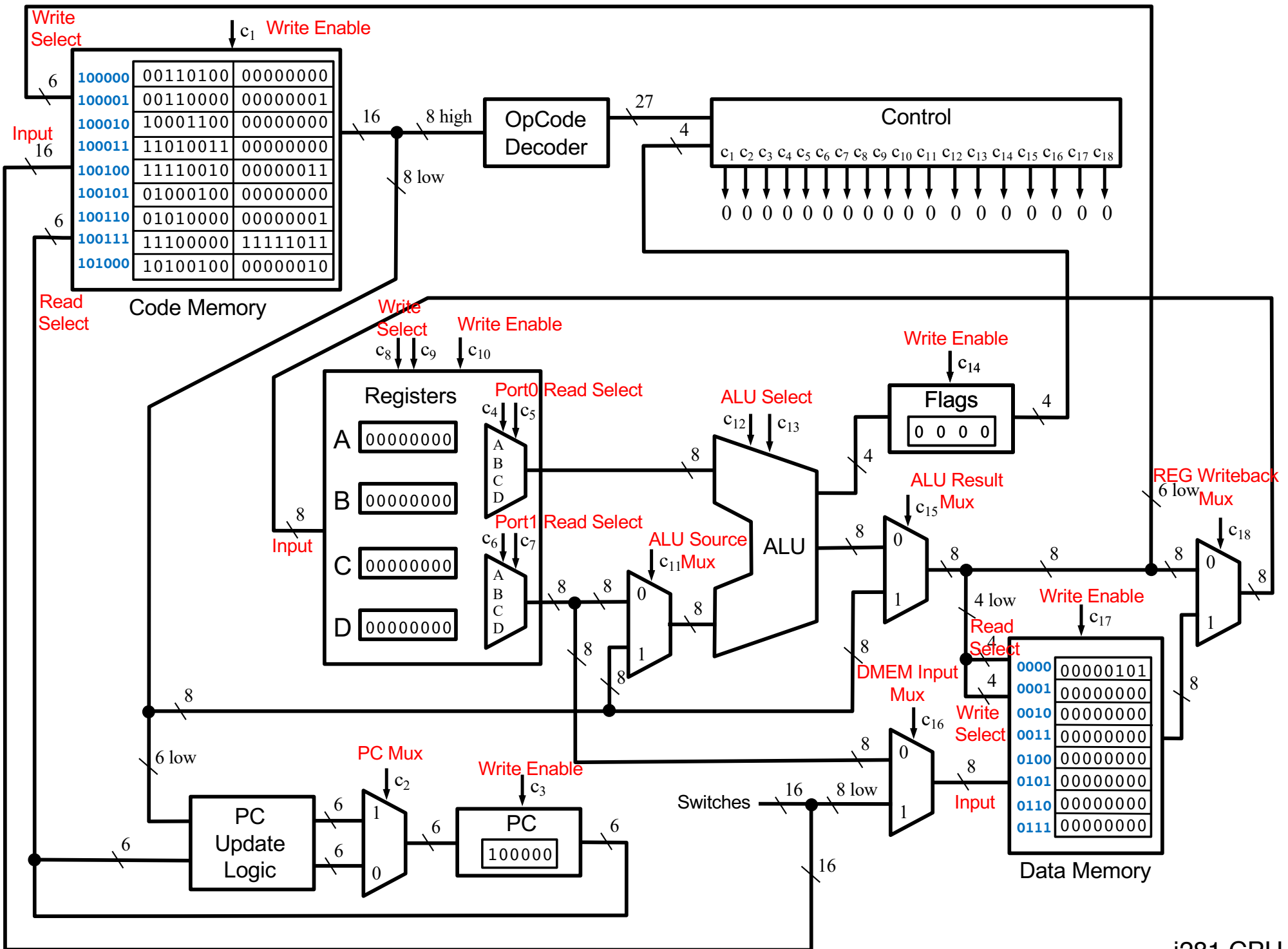
Comments

0000:1000		org	1000h	; Start at 0000:1000h
0000:1000		_memcpy	proc	
0000:1000	55		push	bp
0000:1001	89 E5		mov	bp,sp
0000:1003	06		push	es
0000:1004	8B 4E 06		mov	cx,[bp+6]
0000:1007	E3 11		jcxz	done
0000:1009	8B 76 04		mov	si,[bp+4]
0000:100C	8B 7E 02		mov	di,[bp+2]
0000:100F	1E		push	ds
0000:1010	07		pop	es
0000:1011	8A 04	loop	mov	al,[si]
0000:1013	88 05		mov	[di],al
0000:1015	46		inc	si
0000:1016	47		inc	di
0000:1017	49		dec	cx
0000:1018	75 F7		jnz	loop
0000:101A	07	done	pop	es
0000:101B	5D		pop	bp
0000:101C	29 C0		sub	ax,ax
0000:101E	C3		ret	
0000:101F			end proc	

i281 CPU Architecture







Memory Layout

- **The i281 CPU uses two different memories**

Memory Layout

- **The i281 CPU uses two different memories**
- **Data Memory**
 - 16 x 8 bits
- **Code Memory**
 - 64 x 16 bits

Memory Layout

- The i281 CPU uses two different memories
- Data Memory
 - 16 x 8 bits
- Code Memory
 - 64 x 16 bits
- Note that they have different number of bits

Memory Layout

- The i281 CPU uses two different memories
- Data Memory
 - 16 x 8 bits (only 16 bytes!)
- Code Memory
 - 64 x 16 bits (only 128 bytes!)
- This is a combined total of 144 bytes!

Memory Layout

- The i281 CPU uses two different memories
- Data Memory
 - 16 x 8 bits (only 16 bytes!)
- Code Memory
 - 64 x 16 bits (only 128 bytes!)
- This is a combined total of 144 bytes!
- Which is enough to represent 48 pixels on this slide!

Data Memory Layout

- **Organized as one contiguous block**
- **Data Memory**
 - **Random access**
 - **Read/write memory**
 - **16 x 8 bits**
 - **Only 16 bytes!**

Data Memory Layout

- **Organized as one contiguous block**
- **Data Memory**
 - **Random access**
 - **Read/write memory**
 - **16 x 8 bits**
 - **Only 16 bytes!**
- **Implemented as a register file with 16 registers, each of which is 8-bits wide.**
- **The register file has one read port and one write port. It also has a write enable input.**

Video Card

- **Memory-mapped video memory**
- **The first 8 bytes of the data memory are connected to the 7-segment displays on the Altera board**
- **Writing to these memory cells automatically lights up the displays, which use only the 4 least significant bits**
- **In video game mode each LED is controlled separately using a different set of 7-segment decoders & all 8 bits**
- **The contents of the second 8 bytes of the data memory cannot be visualized, but programs can still use them**

Data Memory Contents for the Bubble Sort Program

```
00000111
00000011
00000010
00000001
00000110
00000100
00000101
00001000
00000111
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```

Data Memory Contents for the Bubble Sort Program

Address	Data
0000	00000111
0001	00000011
0010	00000010
0011	00000001
0100	00000110
0101	00000100
0110	00000101
0111	00001000
1000	00000111
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

Data Memory Contents for the Bubble Sort Program

Address	Data	Comment
0000	00000111	//array[0]
0001	00000011	//array[1]
0010	00000010	//array[2]
0011	00000001	//array[3]
0100	00000110	//array[4]
0101	00000100	//array[5]
0110	00000101	//array[6]
0111	00001000	//array[7]
1000	00000111	//last
1001	00000000	//temp
1010	00000000	
1011	00000000	
1100	00000000	
1101	00000000	
1110	00000000	
1111	00000000	

Memory-Mapped Video Memory

Address	Data
0000	00000111
0001	00000011
0010	00000010
0011	00000001
0100	00000110
0101	00000100
0110	00000101
0111	00001000
1000	00000111
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

Memory-Mapped Video Memory

Address	Data	
0000	00000111	→ 7
0001	00000011	→ 3
0010	00000010	→ 2
0011	00000001	→ 1
0100	00000110	→ 6
0101	00000100	→ 4
0110	00000101	→ 5
0111	00001000	→ 8
1000	00000111	
1001	00000000	
1010	00000000	
1011	00000000	
1100	00000000	
1101	00000000	
1110	00000000	
1111	00000000	

Memory-Mapped Video Memory

Address	Data
0000	00000111 → 3
0001	00000011 → 3
0010	00000010 → 2
0011	00000001 → 1
0100	00000110 → 6
0101	00000100 → 4
0110	01100101 → 5
0111	00001000 → 8
1000	00000111
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

Changing these bits will not affect what is displayed, but it will affect the program.

Memory-Mapped Video Memory

Address	Data
0000	0 1 000111
0001	000 1 0011
0010	1 0000010
0011	0 1 000001
0100	000 1 0110
0101	1 0000100
0110	0 11 00101
0111	0 1 001000
1000	00000111
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

Changing these bits will not affect what is displayed, but it will affect the program.

Memory-Mapped Video Memory

Address	Data	
0000	00000111	→ 1
0001	00000011	→ 3
0010	00000010	→ 2
0011	00000001	→ 1
0100	00000110	→ 3
0101	00000100	→ 4
0110	00000101	→ 5
0111	00001000	→ 8
1000	00000111	
1001	00000000	
1010	00000000	
1011	00000000	
1100	00000000	
1101	00000000	
1110	00000000	
1111	00000000	

This memory cell is used by the program, but it cannot be visualized on the 7-segment displays.

Memory-Mapped Video Memory









Address	Data	
0000	00000111	→ 1
0001	00000011	→ 3
0010	00000010	→ 2
0011	00000001	→ 1
0100	00000110	→ 3
0101	00000100	→ 4
0110	00000101	→ 5
0111	00001000	→ 8
1000	00000111	
1001	00000000	
1010	00000000	
1011	00000000	
1100	00000000	
1101	00000000	
1110	00000000	
1111	00000000	

All of these cannot be visualized on the 7-segment displays.









Memory-Mapped Video Memory in Video Game Mode

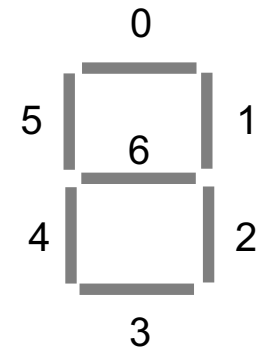
Address	Data
0000	00000000
0001	00000000
0010	00000000
0011	00000000
0100	01111001
0101	01010100
0110	01011110
0111	00000000
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

Memory-Mapped Video Memory in Video Game Mode

Address	Data
0000	00000000 → 
0001	00000000 → 
0010	00000000 → 
0011	00000000 → 
0100	01111001 → 
0101	01010100 → 
0110	01011110 → 
0111	00000000 → 
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

Memory-Mapped Video Memory in Video Game Mode

Address	Data
0000	00000000 → 
0001	00000000 → 
0010	00000000 → 
0011	00000000 → 
0100	01111001 → 
0101	01010100 → 
0110	01011110 → 
0111	00000000 → 
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000



In this case the last 7 bits are used and each controls one of the 7 LEDs.

Code Memory Layout

- **Split into two parts**
- **BIOS Code Memory**
 - Read only memory
 - 32 x 16 bits
- **User Code Memory**
 - Read only memory (in User mode)
 - Read/Write memory (in BIOS mode)
 - 32 x 16 bits

i281 Example:
Add the numbers from 1 to 5

i281 Example:
Add the numbers from 1 to 5

C Language v.s. Assembly Language

C Version

```
// C Version
//
// Add the numbers from 1 to 5 using a for loop.

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++)
        sum+=i;

    // printf("%d\n", sum);
}
```

i281 Assembly Version

.data

```
N      BYTE    5
i      BYTE    ?
sum    BYTE    ?
```

.code

```
      LOADI   B, 0      ; sum=0
      LOADI   A, 1      ; i=1
      LOAD    D, [N]    ; register_D=N
Loop:  CMP     A, D      ; i<=N ?
      BRG     End       ; exit if i>N
Add:   ADD    B, A      ; sum+=i
      ADDI   A, 1      ; i++
      JUMP   Loop       ; next iteration
End:   STORE  [sum], B  ; update the memory for sum
```

; Register allocation:

; A: i

; B: sum

; C: <not used>

; D: N

i281 Assembly Version

.data

```
N      BYTE    5
i      BYTE    ?
sum    BYTE    ?
```

.code

```
      LOADI   B, 0      ; sum=0
      LOADI   A, 1      ; i=1
      LOAD    D, [N]    ; register_D=N
Loop:  CMP     A, D      ; i<=N ?
      BRG     End       ; exit if i>N
Add:   ADD    B, A      ; sum+=i
      ADDI   A, 1      ; i++
      JUMP   Loop       ; next iteration
End:   STORE  [sum], B  ; update the memory for sum
```

; Register allocation:

; A: i

; B: sum

; C: <not used>

; D: N

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N          BYTE    5
i          BYTE    ?
sum        BYTE    ?

.code

        LOADI    B, 0        ; sum=0
        LOADI    A, 1        ; i=1
        LOAD     D, [N]      ; register_D=N
Loop:   CMP      A, D        ; i<=N ?
        BRG      End        ; exit if i>N
Add:    ADD      B, A        ; sum+=i
        ADDI    A, 1        ; i++
        JUMP    Loop        ; next iteration
End:    STORE    [sum], B    ; write B to sum
```


Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N          BYTE    5
i          BYTE    ?
sum        BYTE    ?

.code

        LOADI    B, 0          ; sum=0
        LOADI    A, 1          ; i=1
        LOAD     D, [N]        ; register_D=N
Loop:    CMP      A, D          ; i<=N ?
        BRG      End          ; exit if i>N
Add:     ADD     B, A          ; sum+=i
        ADDI    A, 1          ; i++
        JUMP    Loop          ; next iteration
End:     STORE   [sum], B      ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI  B, 0        ; sum=0
        LOADI  A, 1        ; i=1
        LOAD   D, [N]      ; register_D=N
Loop:   CMP    A, D        ; i<=N ?
        BRG   End        ; exit if i>N
Add:    ADD   B, A        ; sum+=i
        ADDI  A, 1        ; i++
        JUMP  Loop       ; next iteration
End:    STORE [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

i=1

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI  A, 1        ; i=1
        LOAD   D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG    End        ; exit if i>N
Add:    ADD    B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop       ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

This has no analog in the C version,
which is written in a high-level language.

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Load the value of N into register D.

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```


Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD    B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop       ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

i=2

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:    CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:     ADD    B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:     STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI    A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:    CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:     ADD    B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:     STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

i=3

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:    CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:     ADD    B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:     STORE  [sum], B    ; write B to sum
```


Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

i=4

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI    A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

i=5

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE   [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop       ; next iteration
End:    STORE  [sum], B    ; write B to sum
```


Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

i=6

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

Add the numbers from 1 to 5

```
// C Version
// using a for loop

int main()
{
    int N=5;
    int i, sum;

    sum=0;
    for(i=1; i<=N; i++) {
        sum+=i;
    }

    // printf("%d\n", sum);
}
```

```
; Assembly Version

.data
N        BYTE    5
i        BYTE    ?
sum      BYTE    ?

.code

        LOADI   B, 0        ; sum=0
        LOADI   A, 1        ; i=1
        LOAD    D, [N]      ; register_D=N
Loop:   CMP     A, D        ; i<=N ?
        BRG     End        ; exit if i>N
Add:    ADD     B, A        ; sum+=i
        ADDI   A, 1        ; i++
        JUMP   Loop        ; next iteration
End:    STORE  [sum], B    ; write B to sum
```

**i281 Example:
Add the numbers from 1 to 5**

Assembly Language v.s. Machine Language

i281 Assembly Code

.data

```
N          BYTE    5
i          BYTE    ?
sum       BYTE    ?
```

.code

```
          LOADI  B, 0          ; sum=0
          LOADI  A, 1          ; i=1
          LOAD   D, [N]        ; register_D=N
Loop:    CMP    A, D          ; i<=N ?
          BRG    End           ; exit if i>N
Add:    ADD    B, A          ; sum+=i
          ADDI   A, 1          ; i++
          JUMP   Loop         ; next iteration
End:    STORE  [sum], B     ; update the memory for sum
```

i281 Assembly Code

.data

N BYTE 5

i BYTE ?

sum BYTE ?

.code

LOADI B, 0

LOADI A, 1

LOAD D, [N]

Loop: CMP A, D

BRG End

Add: ADD B, A

ADDI A, 1

JUMP Loop

End: STORE [sum], B

Mapping Assembly to Machine Code

.data

```
N      BYTE  5
i      BYTE  ?
sum    BYTE  ?
```

Data Memory:

```
00000101
00000000
00000000
```

.code

```
      LOADI  B, 0
      LOADI  A, 1
      LOAD   D, [N]
Loop:  CMP    A, D
      BRG    End
Add:   ADD   B, A
      ADDI  A, 1
      JUMP  Loop
End:   STORE [sum], B
```

Code Memory:

```
0011010000000000
0011000000000001
1000110000000000
1101001100000000
1111001000000011
0100010000000000
0101000000000001
1110000011111011
1010010000000010
```

Assembly Language

Machine Language

Mapping Assembly to Machine Code

.data

```
N      BYTE  5
i      BYTE  ?
sum    BYTE  ?
```

Data Memory:

```
0000 0101
0000 0000
0000 0000
```

.code

```
      LOADI  B, 0
      LOADI  A, 1
      LOAD   D, [N]
Loop:  CMP    A, D
      BRG    End
Add:   ADD   B, A
      ADDI  A, 1
      JUMP  Loop
End:   STORE [sum], B
```

Code Memory:

```
0011 0100 0000 0000
0011 0000 0000 0001
1000 1100 0000 0000
1101 0011 0000 0000
1111 0010 0000 0011
0100 0100 0000 0000
0101 0000 0000 0001
1110 0000 1111 1011
1010 0100 0000 0010
```

Assembly Language

Machine Language
in Binary

Mapping Assembly to Machine Code

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

Data Memory:

0 **5**
0 **0**
0 **0**

.code

LOADI **B, 0**
 LOADI **A, 1**
 LOAD **D, [N]**
Loop: **CMP** **A, D**
 BRG **End**
Add: **ADD** **B, A**
 ADDI **A, 1**
 JUMP **Loop**
End: **STORE** **[sum], B**

Code Memory:

3 **4** **0** **0**
3 **0** **0** **1**
8 **C** **0** **0**
D **3** **0** **0**
F **2** **0** **3**
4 **4** **0** **0**
5 **0** **0** **1**
E **0** **F** **B**
A **4** **0** **2**

Assembly Language

Machine Language
in Binary

Mapping Assembly to Machine Code

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

Data Memory:

05
00
00

.code

LOADI **B, 0**
 LOADI **A, 1**
 LOAD **D, [N]**
Loop: **CMP** **A, D**
 BRG **End**
Add: **ADD** **B, A**
 ADDI **A, 1**
 JUMP **Loop**
End: **STORE** **[sum], B**

Code Memory:

34 00
30 01
8C 00
D3 00
F2 03
44 00
50 01
E0 FB
A4 02

Assembly Language

Machine Language
in Hexadecimal

i281 Example:
Add the numbers from 1 to 5

Preview of OPCODEs

Mapping Assembly to Machine Code

.data

```
N      BYTE  5
i      BYTE  ?
sum    BYTE  ?
```

Data Memory:

```
00000101
00000000
00000000
```

.code

```
      LOADI  B, 0
      LOADI  A, 1
      LOAD   D, [N]
Loop:  CMP    A, D
      BRG    End
Add:   ADD   B, A
      ADDI  A, 1
      JUMP  Loop
End:   STORE [sum], B
```

Code Memory:

```
0011010000000000
0011000000000001
1000110000000000
1101001100000000
1111001000000011
0100010000000000
0101000000000001
1110000011111011
1010010000000010
```

Assembly Language

Machine Language

Mapping Assembly to Machine Code

.data

```
N      BYTE  5
i      BYTE  ?
sum    BYTE  ?
```

Data Memory:

```
00000101
00000000
00000000
```

.code

```
      LOADI  B, 0
      LOADI  A, 1
      LOAD   D, [N]
Loop:  CMP    A, D
      BRG    End
Add:   ADD   B, A
      ADDI  A, 1
      JUMP  Loop
End:   STORE [sum], B
```

Code Memory:

```
00110100_00000000
00110000_00000001
10001100_00000000
11010011_00000000
11110010_00000011
01000100_00000000
01010000_00000001
11100000_11111011
10100100_00000010
```

Assembly Language

Machine Language

Mapping Assembly to Machine Code

.data

```
N      BYTE  5
i      BYTE  ?
sum    BYTE  ?
```

Data Memory:

```
00000101
00000000
00000000
```

.code

```
      LOADI  B, 0
      LOADI  A, 1
      LOAD   D, [N]
Loop:  CMP    A, D
      BRG    End
Add:   ADD   B, A
      ADDI  A, 1
      JUMP  Loop
End:   STORE [sum], B
```

Code Memory:

```
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

Assembly Language

Machine Language

Mapping Assembly to Machine Code

.data

```
N      BYTE  5
i      BYTE  ?
sum    BYTE  ?
```

Data Memory:

```
00000101
00000000
00000000
```

.code

```
      LOADI  B, 0
      LOADI  A, 1
      LOAD   D, [N]
Loop:  CMP    A, D
      BRG    End
Add:   ADD   B, A
      ADDI  A, 1
      JUMP  Loop
End:   STORE [sum], B
```

Code Memory:

```
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

Mapping Assembly to Machine Code

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

Data Memory:

00000101
00000000
00000000

.code

LOADI **B, 0**
 LOADI **A, 1**
 LOAD **D, [N]**
Loop: **CMP** **A, D**
 BRG **End**
Add: **ADD** **B, A**
 ADDI **A, 1**
 JUMP **Loop**
End: **STORE** **[sum], B**

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

Mapping Assembly to Machine Code

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

Data Memory:

00000101
00000000
00000000

.code

LOADI **B, 0**
 LOADI **A, 1**
 LOAD **D, [N]**
Loop: **CMP** **A, D**
 BRG **End**
Add: **ADD** **B, A**
 ADDI **A, 1**
 JUMP **Loop**
End: **STORE** **[sum], B**

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

Mapping Assembly to Machine Code

.data

```
N      BYTE  5
i      BYTE  ?
sum    BYTE  ?
```

Data Memory:

```
00000101
00000000
00000000
```

.code

```
      LOADI  B, 0
      LOADI  A, 1
      LOAD   D, [N]
Loop:  CMP    A, D
      BRG    End
Add:   ADD   B, A
      ADDI  A, 1
      JUMP  Loop
End:   STORE [sum], B
```

Code Memory:

```
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

OPCODE Mapping

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

Data Memory:

00000101
00000000
00000000

.code

LOADI **B, 0**
 LOADI **A, 1**
 LOAD **D, [N]**
Loop: **CMP** **A, D**
 BRG **End**
Add: **ADD** **B, A**
 ADDI **A, 1**
 JUMP **Loop**
End: **STORE** **[sum], B**

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

OPCODE Mapping

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

Data Memory:

00000101
00000000
00000000

.code

LOADI **B, 0**
 LOADI **A, 1**
 LOAD **D, [N]**
Loop: **CMP** **A, D**
 BRG **End**
Add: **ADD** **B, A**
 ADDI **A, 1**
 JUMP **Loop**
End: **STORE** **[sum], B**

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

Register Parameter Mapping

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

Data Memory:

00000101
00000000
00000000

.code

LOADI **B**, **0**
 LOADI **A**, **1**
 LOAD **D**, [**N**]
Loop: **CMP** **A**, **D**
 BRG **End**
Add: **ADD** **B**, **A**
 ADDI **A**, **1**
 JUMP **Loop**
End: **STORE** [**sum**], **B**

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

Register Parameter Mapping

.data

N	BYTE	5
i	BYTE	?
sum	BYTE	?

Data Memory:

00000101
00000000
00000000

.code

	LOADI	B, 0
	LOADI	A, 1
	LOAD	D, [N]
Loop:	CMP	A, D
	BRG	End
Add:	ADD	B, A
	ADDI	A, 1
	JUMP	Loop
End:	STORE	[sum], B

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

Second Register Parameter Mapping

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

Data Memory:

00000101
00000000
00000000

.code

LOADI **B**, **0**
 LOADI **A**, **1**
 LOAD **D**, **[N]**
Loop: **CMP** **A**, **D**
 BRG **End**
Add: **ADD** **B**, **A**
 ADDI **A**, **1**
 JUMP **Loop**
End: **STORE** **[sum]**, **B**

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

Second Register Parameter Mapping

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

Data Memory:

00000101
00000000
00000000

.code

LOADI **B**, **0**
 LOADI **A**, **1**
 LOAD **D**, [**N**]
Loop: **CMP** **A**, **D**
 BRG **End**
Add: **ADD** **B**, **A**
 ADDI **A**, **1**
 JUMP **Loop**
End: **STORE** [**sum**], **B**

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

Value / Address / Offset Mapping

.data

N	BYTE	5
i	BYTE	?
sum	BYTE	?

Data Memory:

00000101
00000000
00000000

.code

	LOADI	B, 0
	LOADI	A, 1
	LOAD	D, [N]
Loop:	CMP	A, D
	BRG	End
Add:	ADD	B, A
	ADDI	A, 1
	JUMP	Loop
End:	STORE	[sum], B

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

Value / Address / Offset Mapping

.data

N	BYTE	5
i	BYTE	?
sum	BYTE	?

Data Memory:

00000101
00000000
00000000

.code

	LOADI	B, 0
	LOADI	A, 1
	LOAD	D, [N]
Loop:	CMP	A, D
	BRG	End
Add:	ADD	B, A
	ADDI	A, 1
	JUMP	Loop
End:	STORE	[sum], B

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

“Don’t care” bits ...

.data

```
N      BYTE  5
i      BYTE  ?
sum    BYTE  ?
```

Data Memory:

```
00000101
00000000
00000000
```

.code

```
      LOADI  B, 0
      LOADI  A, 1
      LOAD   D, [N]
Loop:  CMP    A, D
      BRG    End
Add:   ADD   B, A
      ADDI  A, 1
      JUMP  Loop
End:   STORE [sum], B
```

Code Memory:

```
0011_01_dd_00000000
0011_00_dd_00000001
1000_11_dd_00000000
1101_00_11_dddddddd
1111_dd_10_00000011
0100_01_00_dddddddd
0101_00_dd_00000001
1110_dd_dd_11110111
1010_01_dd_00000010
```

... are mapped to 0 by the Assembler

.data

```
N      BYTE  5
i      BYTE  ?
sum    BYTE  ?
```

Data Memory:

```
00000101
00000000
00000000
```

.code

```
      LOADI B, 0
      LOADI A, 1
      LOAD  D, [N]
Loop: CMP   A, D
      BRG   End
Add: ADD   B, A
      ADDI  A, 1
      JUMP  Loop
End:  STORE [sum], B
```

Code Memory:

```
0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010
```

Mapping Assembly to Machine Code

.data

N **BYTE** **5**
i **BYTE** **?**
sum **BYTE** **?**

Data Memory:

00000101
00000000
00000000

.code

LOADI **B, 0**
 LOADI **A, 1**
 LOAD **D, [N]**
Loop: **CMP** **A, D**
 BRG **End**
Add: **ADD** **B, A**
 ADDI **A, 1**
 JUMP **Loop**
End: **STORE** **[sum], B**

Code Memory:

0011_01_00_00000000
0011_00_00_00000001
1000_11_00_00000000
1101_00_11_00000000
1111_00_10_00000011
0100_01_00_00000000
0101_00_00_00000001
1110_00_00_11111011
1010_01_00_00000010

The Assembly Language Instructions

The i281 Assembly Instructions

NOOP	NO OPERATION
INPUTC	INPUT into Code memory
INPUTCF	INPUT into Code memory with offset
INPUTD	INPUT into Data memory
INPUTDF	INPUT into Data memory with offset
MOVE	MOVE the contents of one register into another
LOADI	LOAD Immediate value
LOADP	LOAD Pointer address
ADD	ADD two registers
ADDI	ADD an Immediate value to a register
SUB	SUBtract two registers
SUBI	SUBtract an Immediate value from a register
LOAD	LOAD from a data memory address into a register
LOADF	LOAD with an offset specified by another register
STORE	STORE a register into a data memory address
STOREF	STORE with an offset specified by another register
SHIFTL	SHIFT Left all bits in a register
SHIFTR	SHIFT Right all bits in a register
CMP	COMPare the values in two registers
JUMP	JUMP unconditionally to a specified address
BRE	BRanch if Equal
BRZ	BRanch if Zero
BRNE	BRanch if Not Equal
BRNZ	BRanch if Not Zero
BRG	BRanch if Greater
BRGE	BRanch if Greater than or Equal

The i281 Assembly Instructions

NOOP	NO OPERATION
INPUTC	INPUT into Code memory
INPUTCF	INPUT into Code memory with offset
INPUTD	INPUT into Data memory
INPUTDF	INPUT into Data memory with offset
MOVE	MOVE the contents of one register into another
LOADI	LOAD Immediate value
LOADP	LOAD Pointer address
ADD	ADD two registers
ADDI	ADD an Immediate value to a register
SUB	SUBtract two registers
SUBI	SUBtract an Immediate value from a register
LOAD	LOAD from a data memory address into a register
LOADF	LOAD with an offset specified by another register
STORE	STORE a register into a data memory address
STOREF	STORE with an offset specified by another register
SHIFTL	SHIFT Left all bits in a register
SHIFTR	SHIFT Right all bits in a register
CMP	COMPARE the values in two registers
JUMP	JUMP unconditionally to a specified address
BRE	BRanch if Equal
BRZ	BRanch if Zero
BRNE	BRanch if Not Equal
BRNZ	BRanch if Not Zero
BRG	BRanch if Greater
BRGE	BRanch if Greater than or Equal

The OPCODEs

There are only 26 OPCODEs

NOOP

INPUTC

INPUTCF

INPUTD

INPUTDF

MOVE

LOADI

LOADP

ADD

ADDI

SUB

SUBI

LOAD

LOADF

STORE

STOREF

SHIFTL

SHIFTR

CMP

JUMP

BRE

BRZ

BRNE

BRNZ

BRG

BRGE

There are only 26 OPCODEs

NOOP	ADD	SHIFTL
INPUTC	ADDI	SHIFTR
INPUTCF	SUB	CMP
INPUTD	SUBI	JUMP
INPUTDF	LOAD	BRE
MOVE	LOADF	BRZ
LOADI	STORE	BRNE
LOADP	STOREF	BRNZ
		BRG
		BRGE

All of these are available in the assembly language for this processor.
However, three pairs are aliased at the machine language level.

There are only ²⁵~~26~~ OPCODEs

NOOP

INPUTC

INPUTCF

INPUTD

INPUTDF

MOVE

LOADI

LOADP

these two
are aliased

ADD

ADDI

SUB

SUBI

LOAD

LOADF

STORE

STOREF

SHIFTL

SHIFTR

CMP

JUMP

BRE

BRZ

BRNE

BRNZ

BRG

BRGE

They have a different meaning in the assembly language, but the assembler maps them to the same machine language OPCODE.

There are only ²⁵~~26~~ OPCODES

NOOP

INPUTC

INPUTCF

INPUTD

INPUTDF

MOVE

LOADI/LOADP

ADD

ADDI

SUB

SUBI

LOAD

LOADF

STORE

STOREF

SHIFTL

SHIFTR

CMP

JUMP

BRE

BRZ

BRNE

BRNZ

BRG

BRGE

There are only ²⁴~~26~~ OPCODES

NOOP
INPUTC
INPUTCF
INPUTD
INPUTDF
MOVE
LOADI/LOADP

ADD
ADDI
SUB
SUBI
LOAD
LOADF
STORE
STOREF

SHIFTL
SHIFTR
CMP
JUMP
BRE
BRZ
BRNE
BRNZ
BRG
BRGE

these two
are aliased

There are only ²³~~26~~ OPCODES

NOOP
INPUTC
INPUTCF
INPUTD
INPUTDF
MOVE
LOADI/LOADP

ADD
ADDI
SUB
SUBI
LOAD
LOADF
STORE
STOREF

SHIFTL
SHIFTR
CMP
JUMP
BRE
BRZ
BRNE
BRNZ
BRG
BRGE

these two
are aliased

these two
are aliased

There are only 23 OPCODES

NOOP

INPUTC

INPUTCF

INPUTD

INPUTDF

MOVE

LOADI/LOADP

ADD

ADDI

SUB

SUBI

LOAD

LOADF

STORE

STOREF

SHIFTL

SHIFTR

CMP

JUMP

BRE/BRZ

BRNE/BRNZ

BRG

BRGE

The OPCODEs

(Mapped to Machine Language)

The OPCODEs

NOOP

0	0	0	0	d	d	d	d	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

INPUTC

0	0	0	1	d	d	0	0	C	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

INPUTCF

0	0	0	1	R	X	0	1	C	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

INPUTD

0	0	0	1	d	d	1	0	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

INPUTDF

0	0	0	1	R	X	1	1	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

MOVE

0	0	1	0	R	X	R	Y	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

LOADI/LOADP

0	0	1	1	R	X	d	d	I	M	M	E	D	V	A	L
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

The OPCODEs

ADD

0	1	0	0	R	X	R	Y	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADDI

0	1	0	1	R	X	d	d	I	M	M	E	D	V	A	L
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

SUB

0	1	1	0	R	X	R	Y	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

SUBI

0	1	1	1	R	X	d	d	I	M	M	E	D	V	A	L
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

LOAD

1	0	0	0	R	X	d	d	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

LOADF

1	0	0	1	R	X	R	Y	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

STORE

1	0	1	0	R	X	d	d	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

STOREF

1	0	1	1	R	X	R	Y	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

The OPCODEs

SHIFTL

1	1	0	0	R	X	d	0	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

SHIFTR

1	1	0	0	R	X	d	1	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

CMP

1	1	0	1	R	X	R	Y	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

JUMP

1	1	1	0	d	d	d	d	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BRE/BRZ

1	1	1	1	d	d	0	0	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BRNE/BRNZ

1	1	1	1	d	d	0	1	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BRG

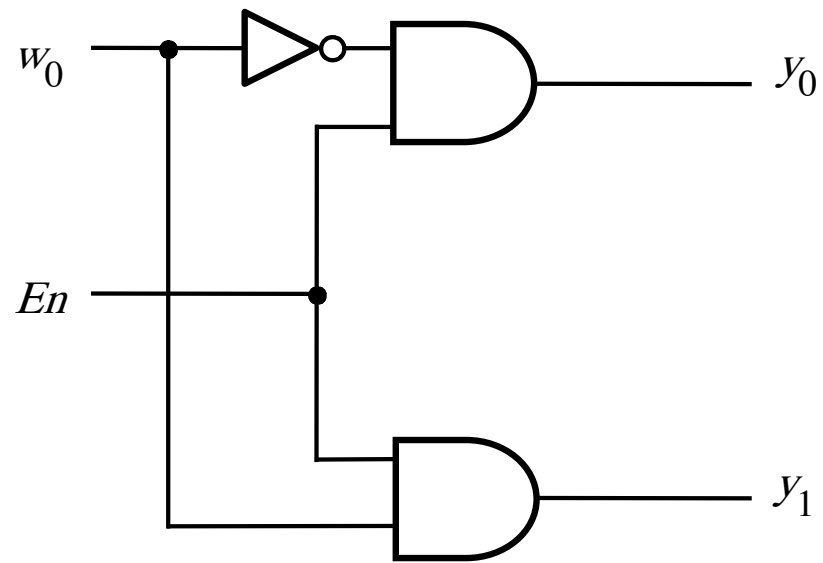
1	1	1	1	d	d	1	0	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BRGE

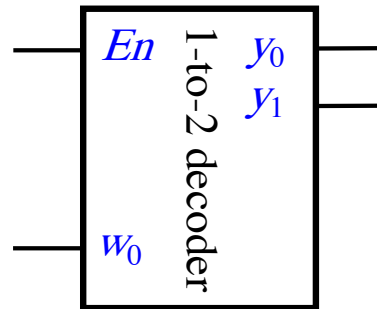
1	1	1	1	d	d	1	1	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Quick Review: Decoders

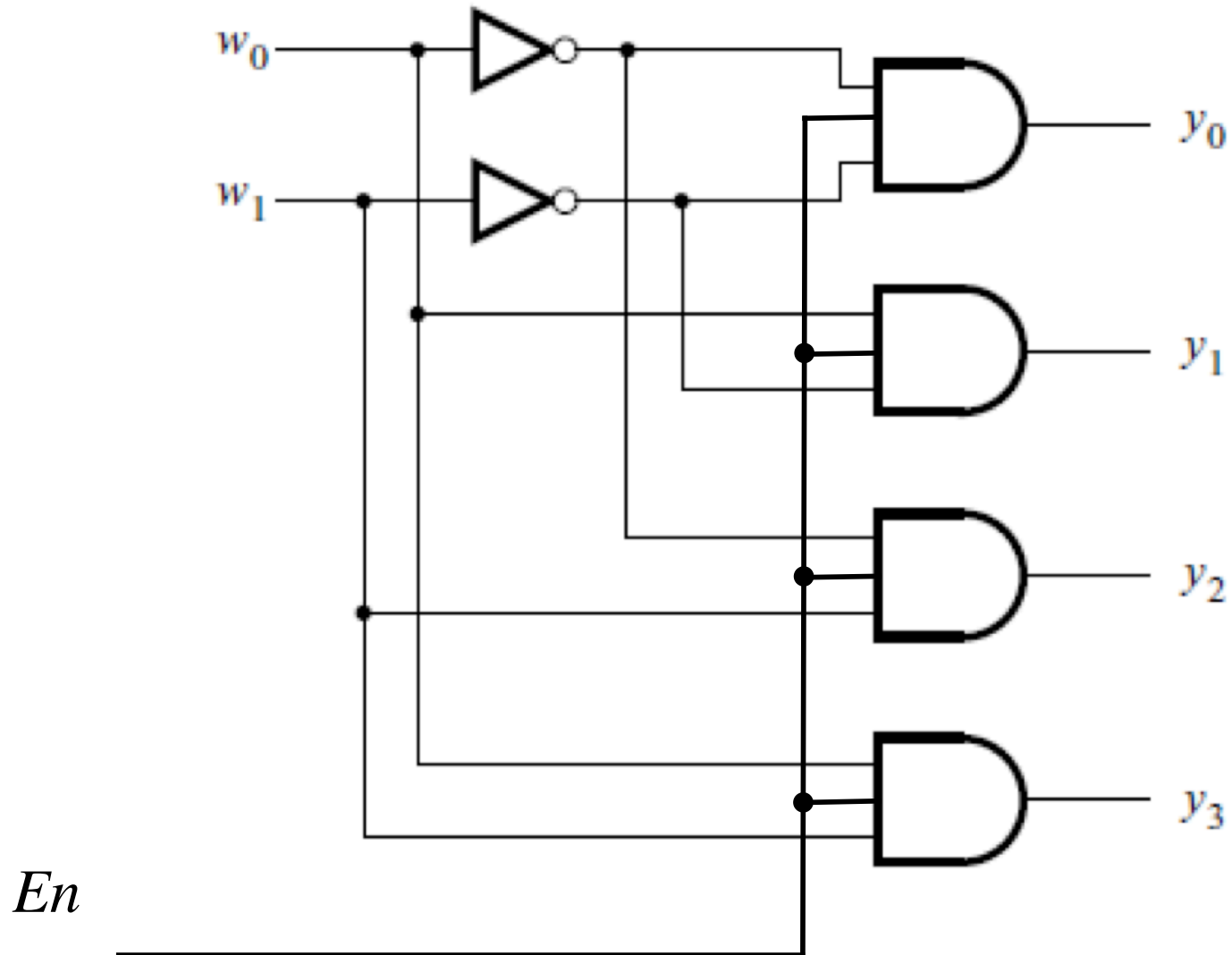
1-to-2 decoder



1-to-2 decoder

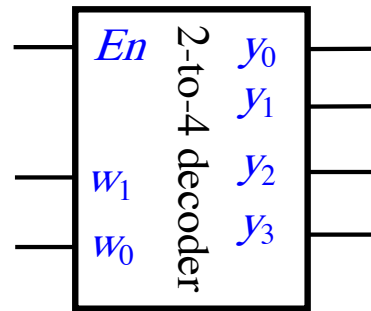


2-to-4 decoder

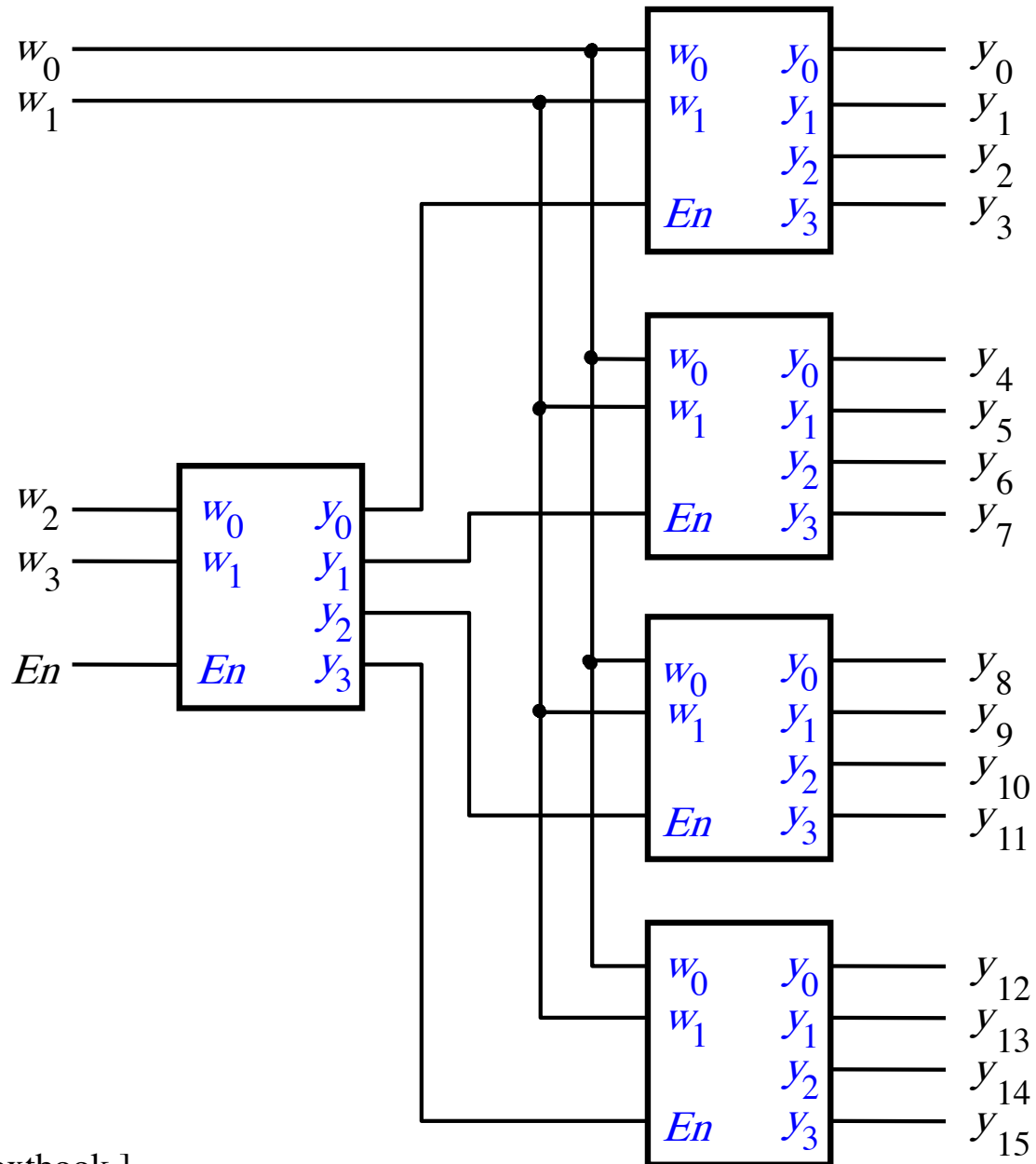


[Figure 4.14c from the textbook]

2-to-4 decoder

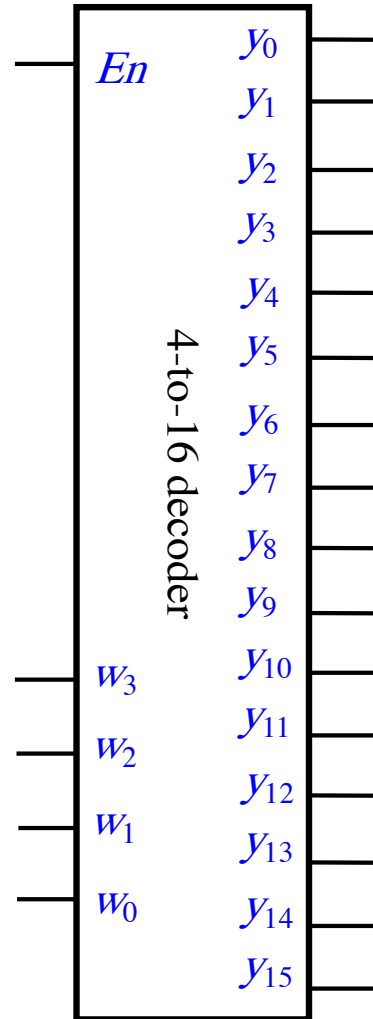


4-to-16 decoder built using a decoder tree

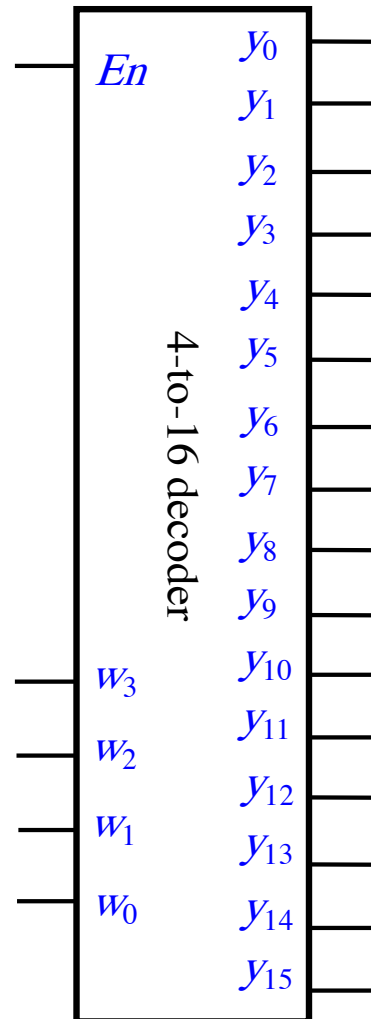


[Figure 4.16 from the textbook]

4-to-16 decoder

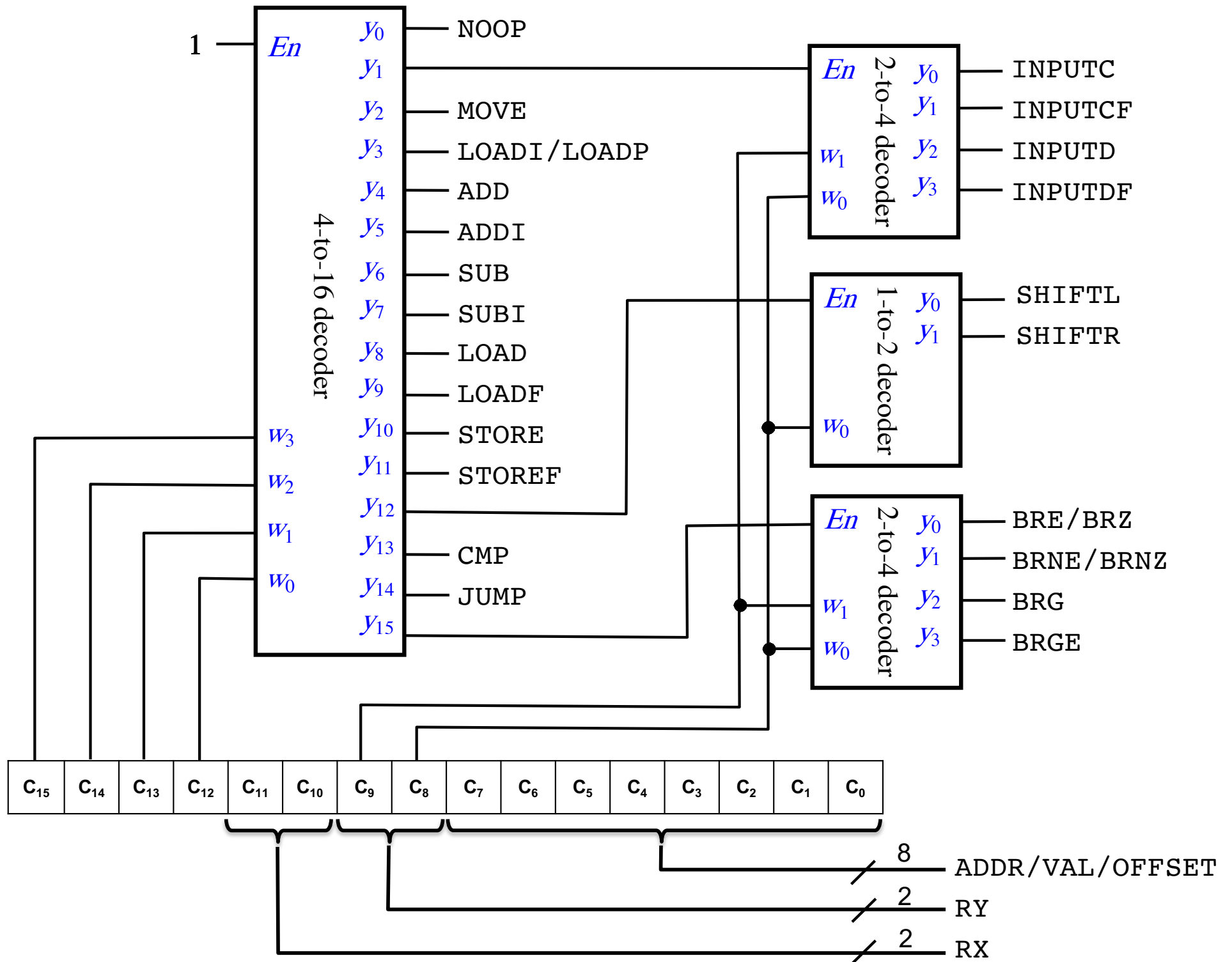


4-to-16 decoder

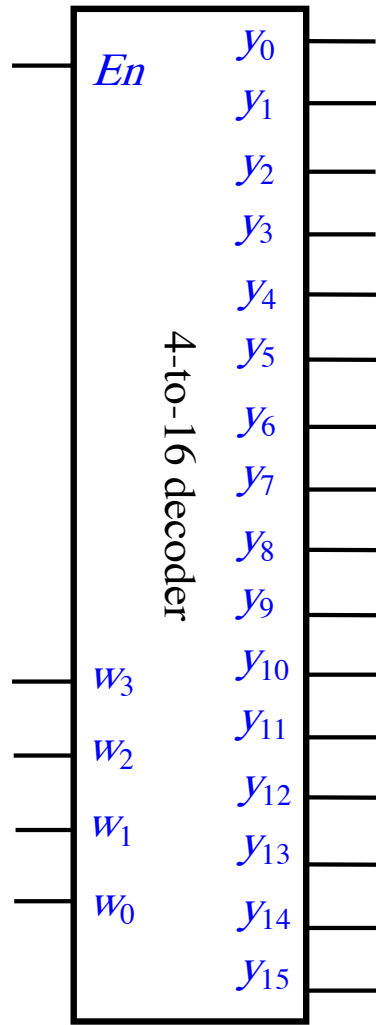


The outputs are one-hot encoded when $En=1$

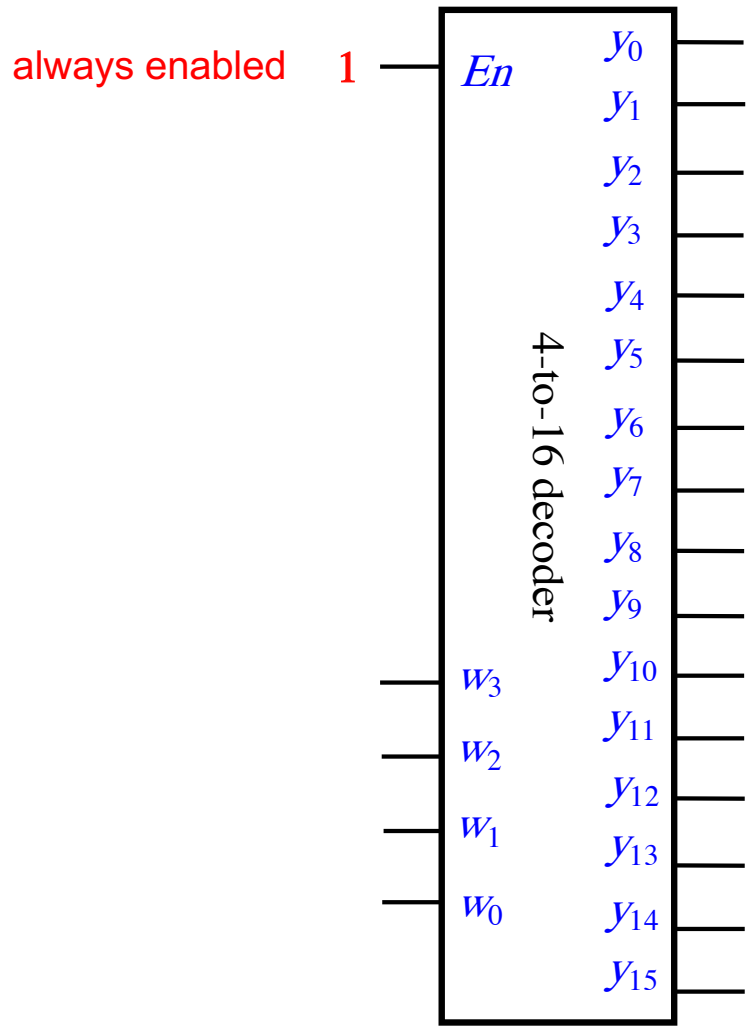
OPCODE Decoding Circuit



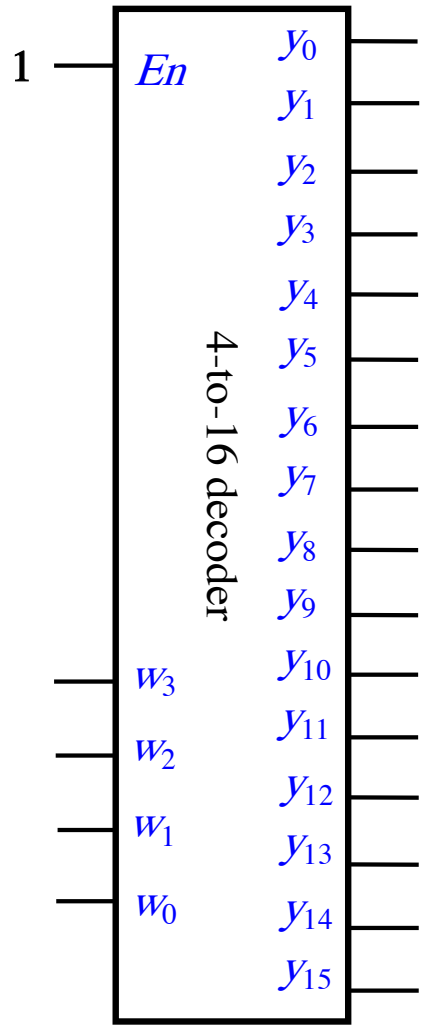
C_{15}	C_{14}	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
----------	----------	----------	----------	----------	----------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------



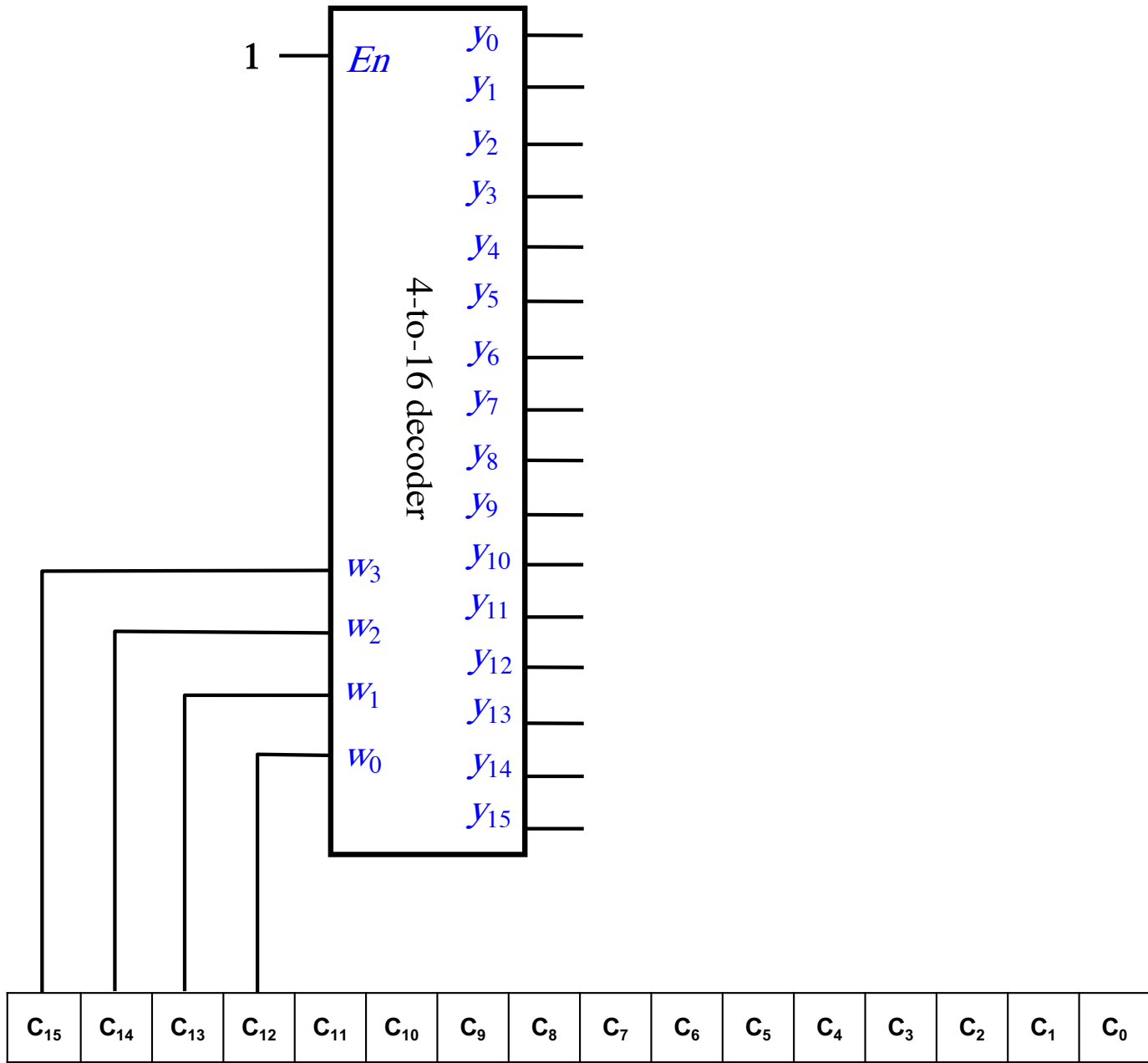
C_{15}	C_{14}	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
----------	----------	----------	----------	----------	----------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

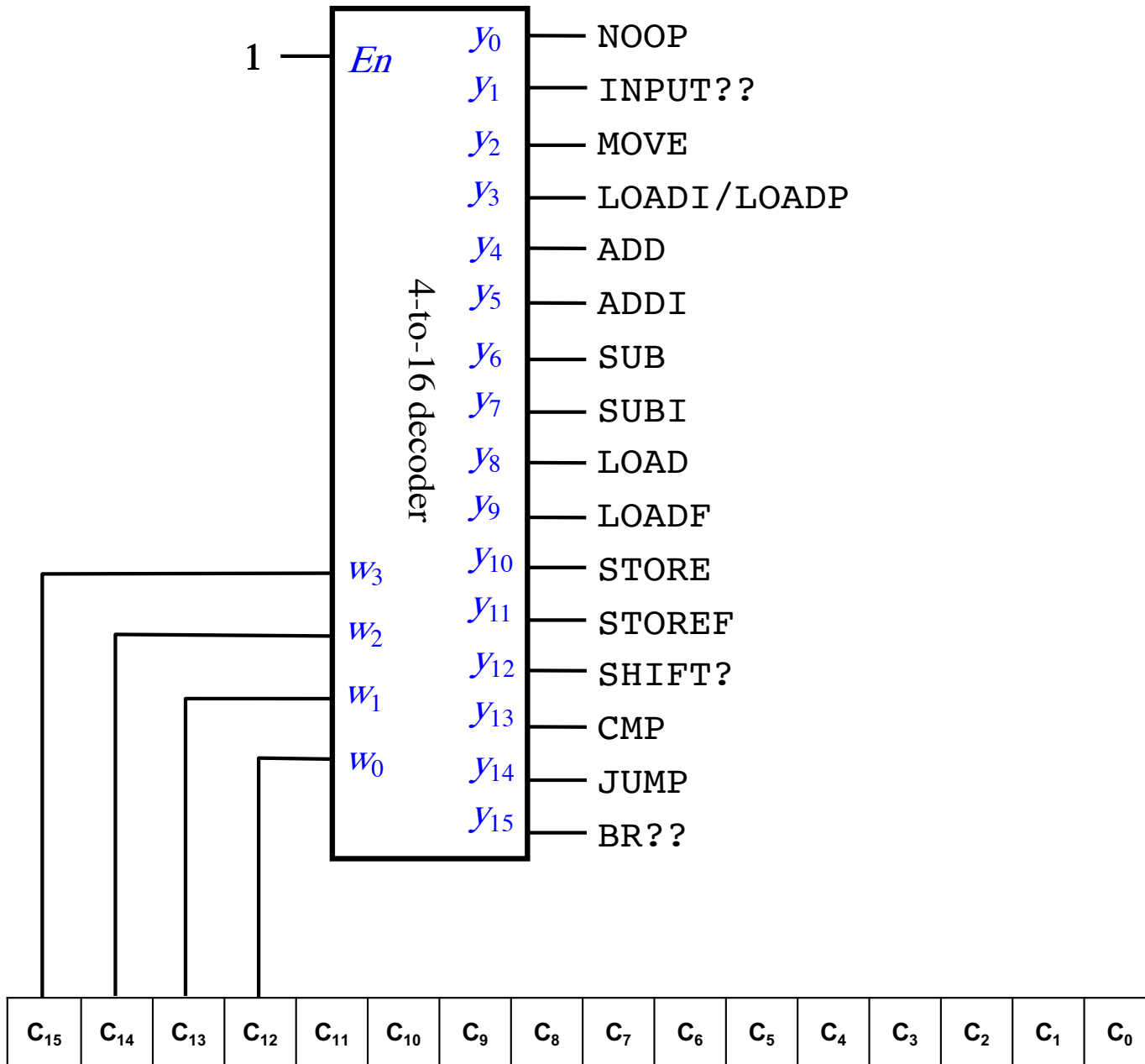


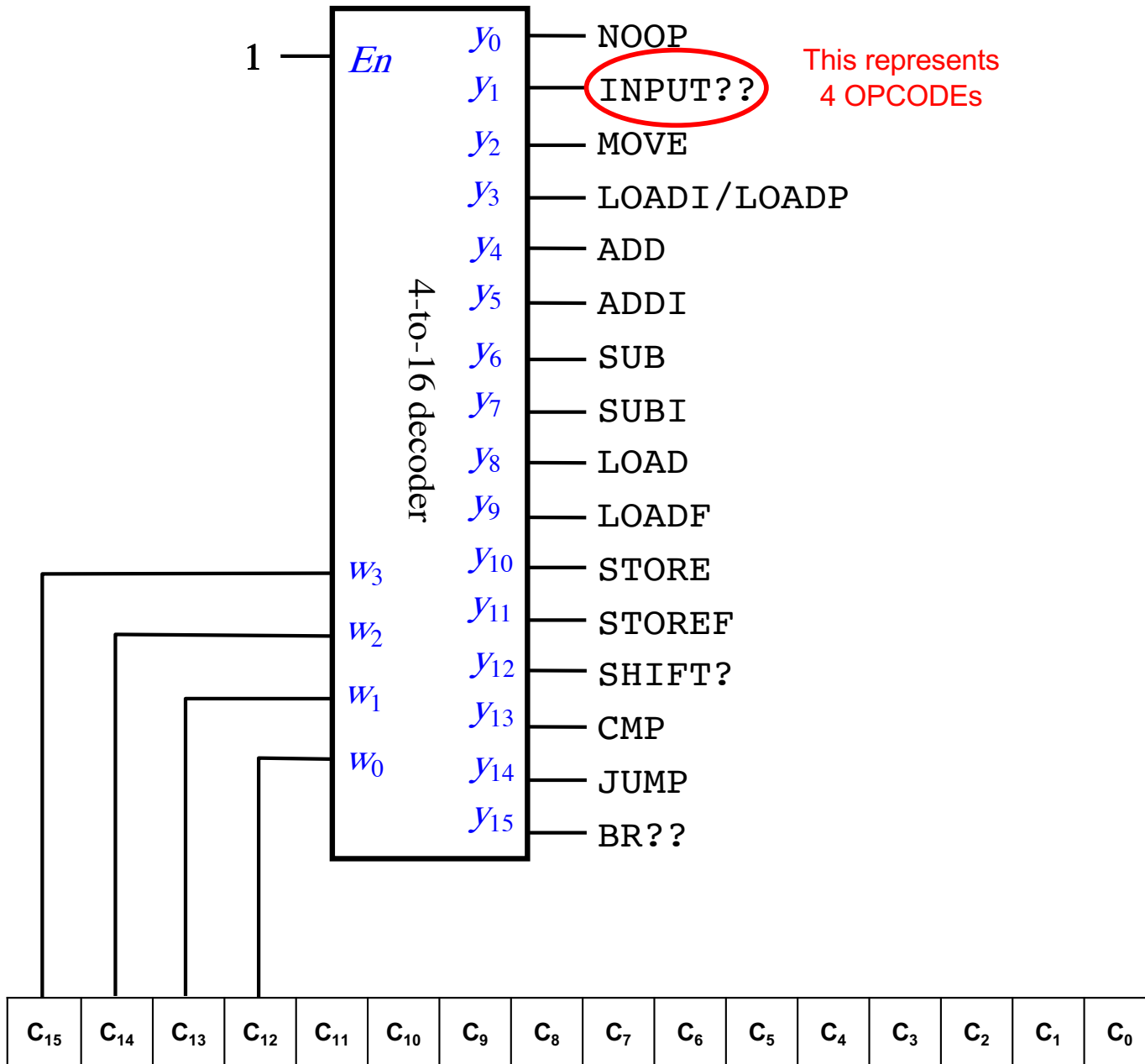
C_{15}	C_{14}	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
----------	----------	----------	----------	----------	----------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

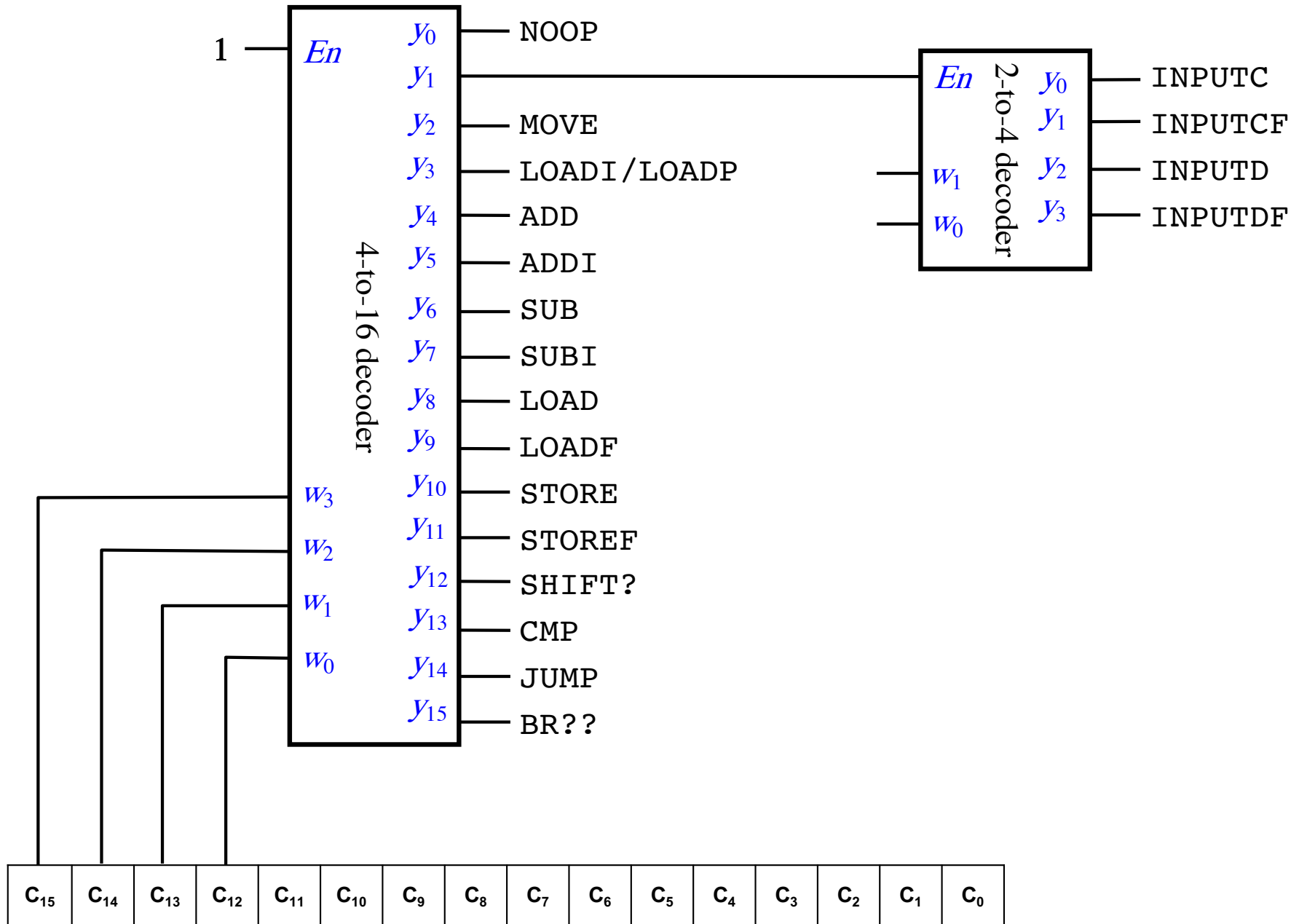


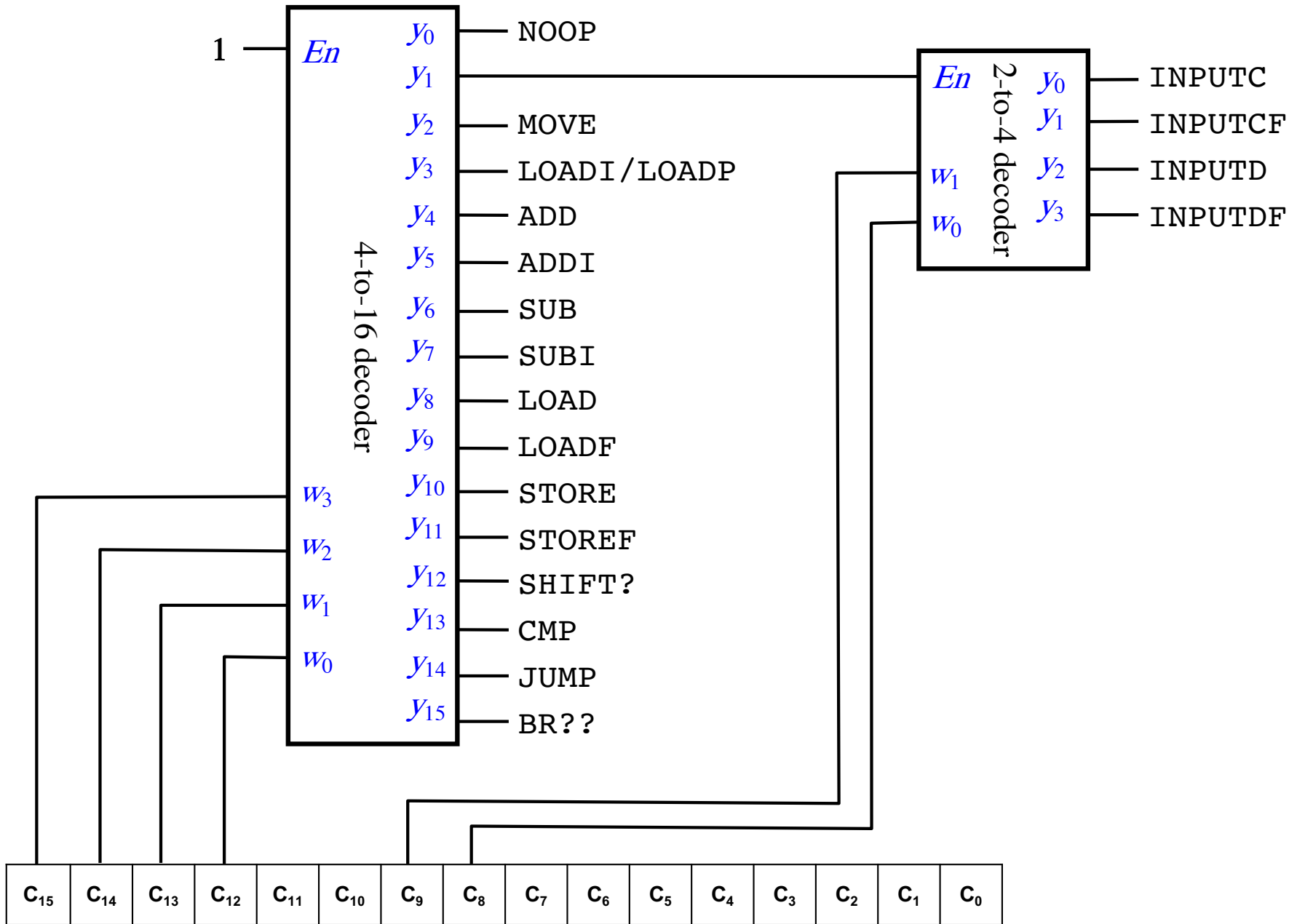
C₁₅	C₁₄	C₁₃	C₁₂	C₁₁	C₁₀	C₉	C₈	C₇	C₆	C₅	C₄	C₃	C₂	C₁	C₀
-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	-----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------

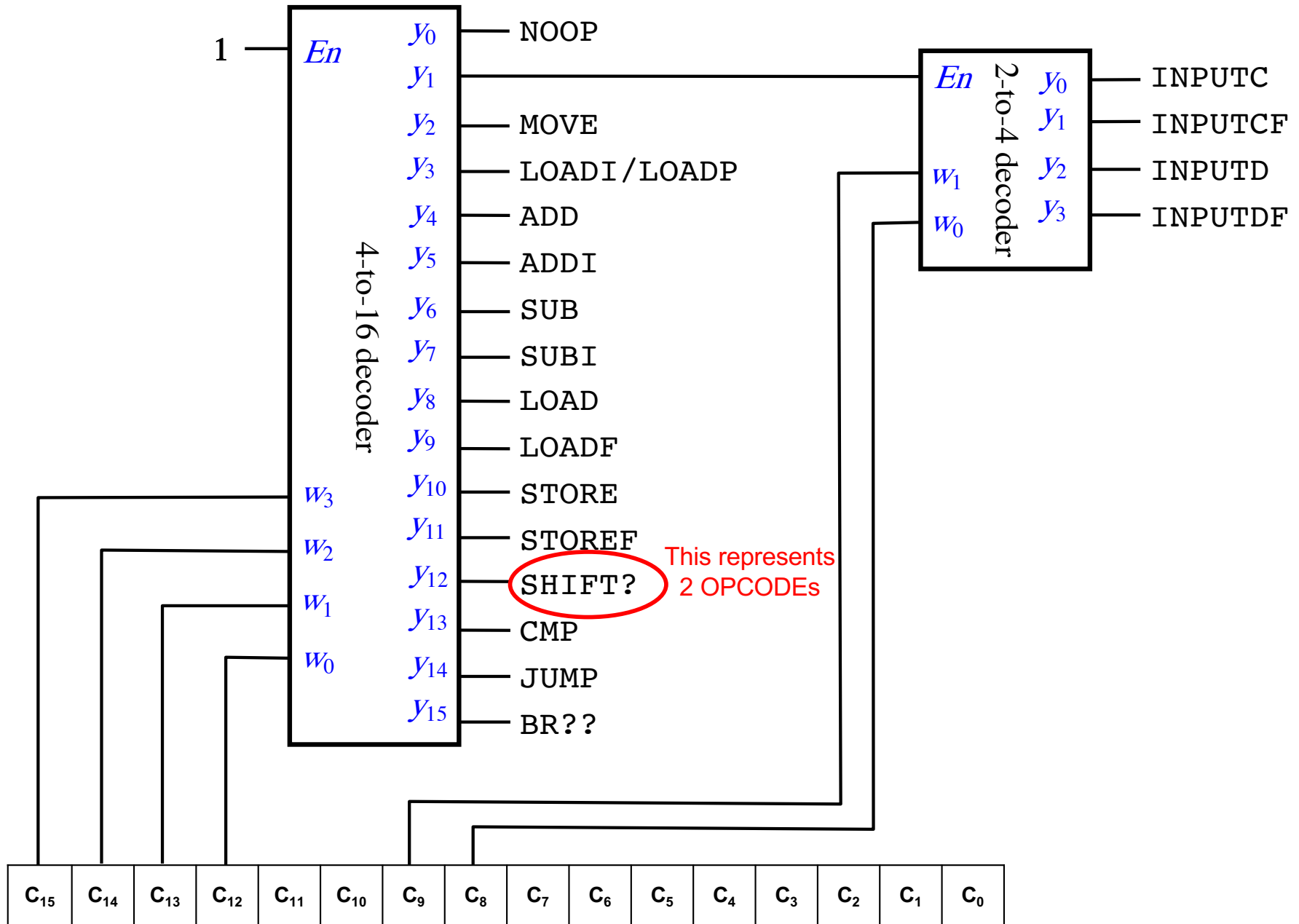


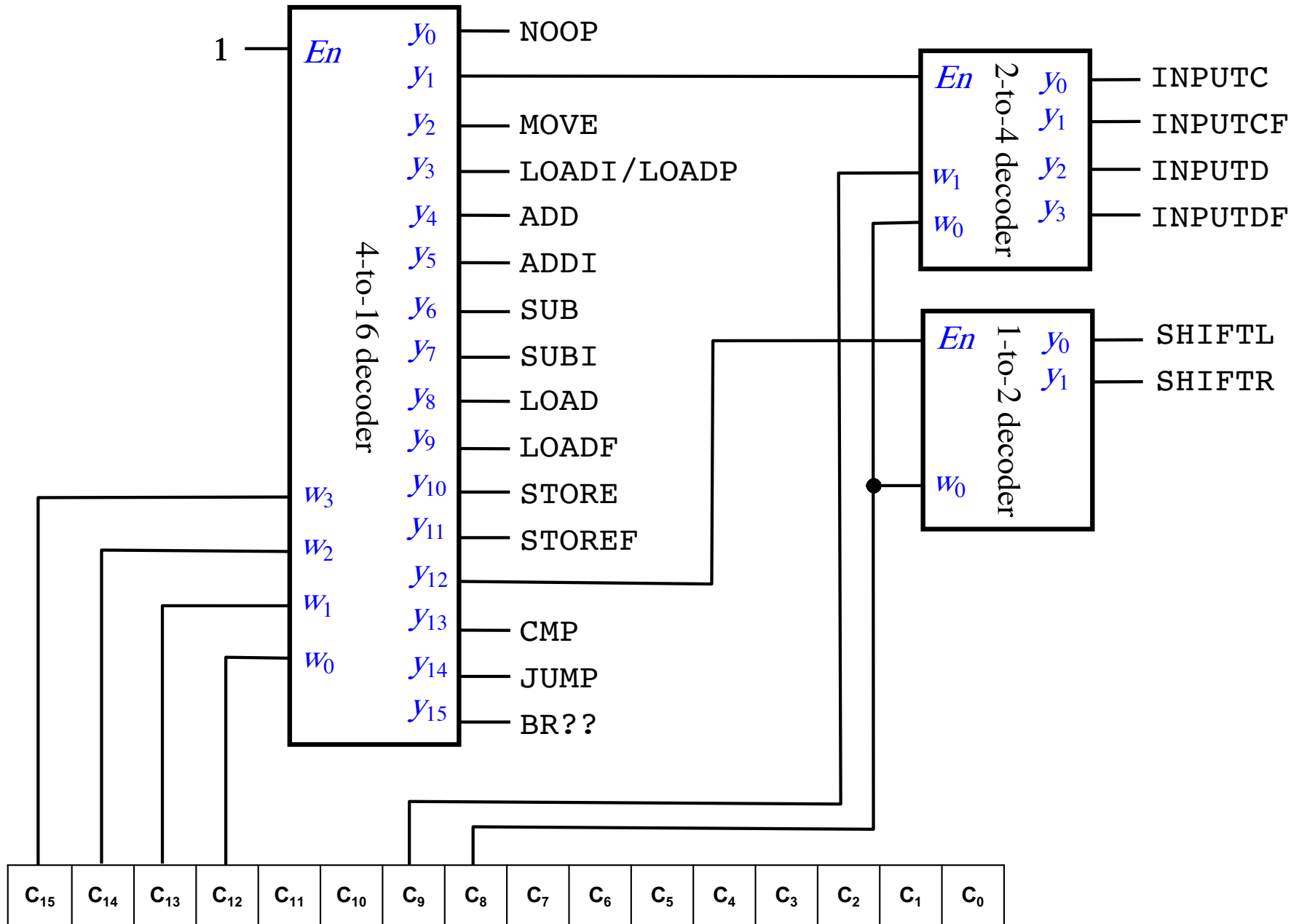


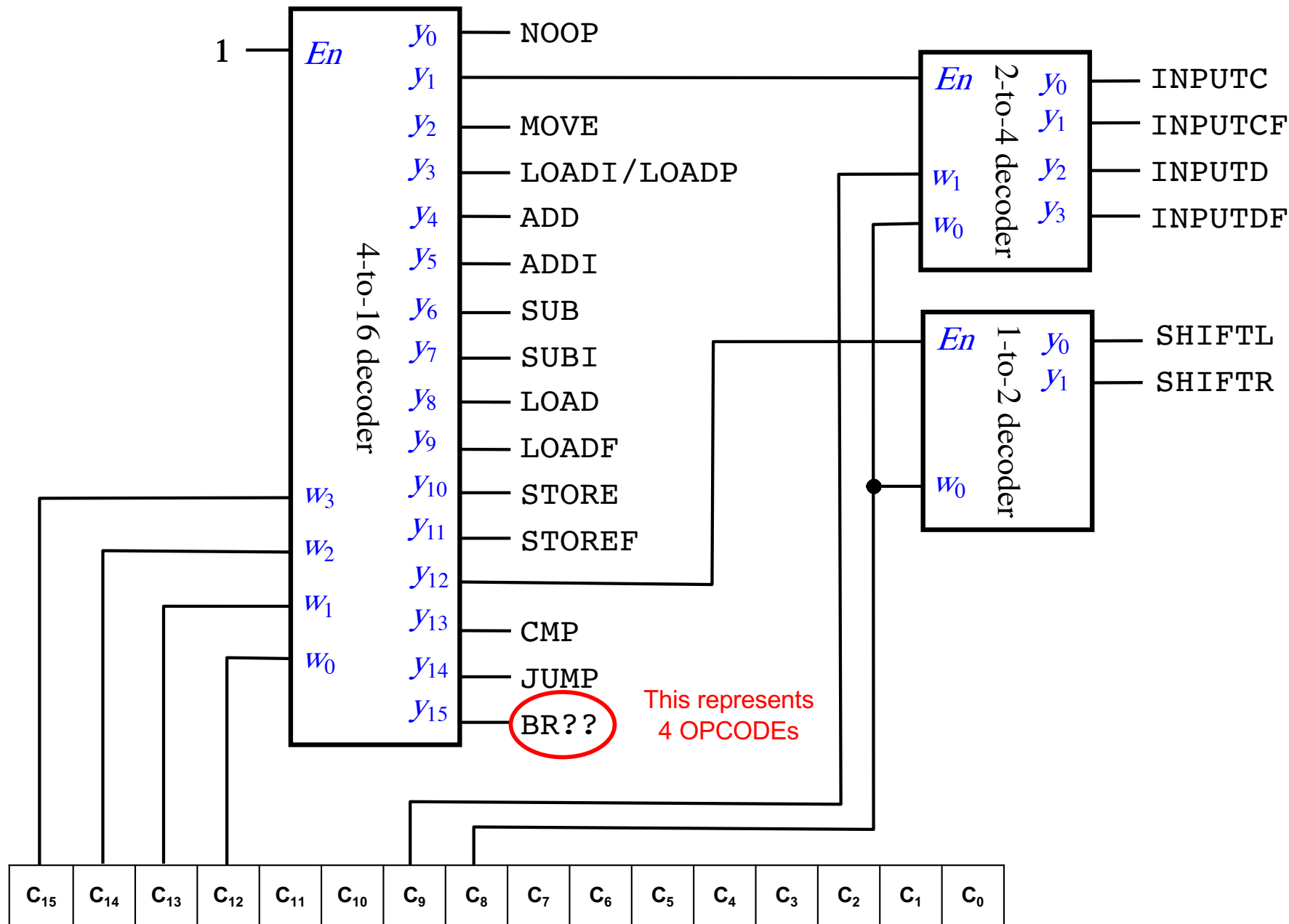


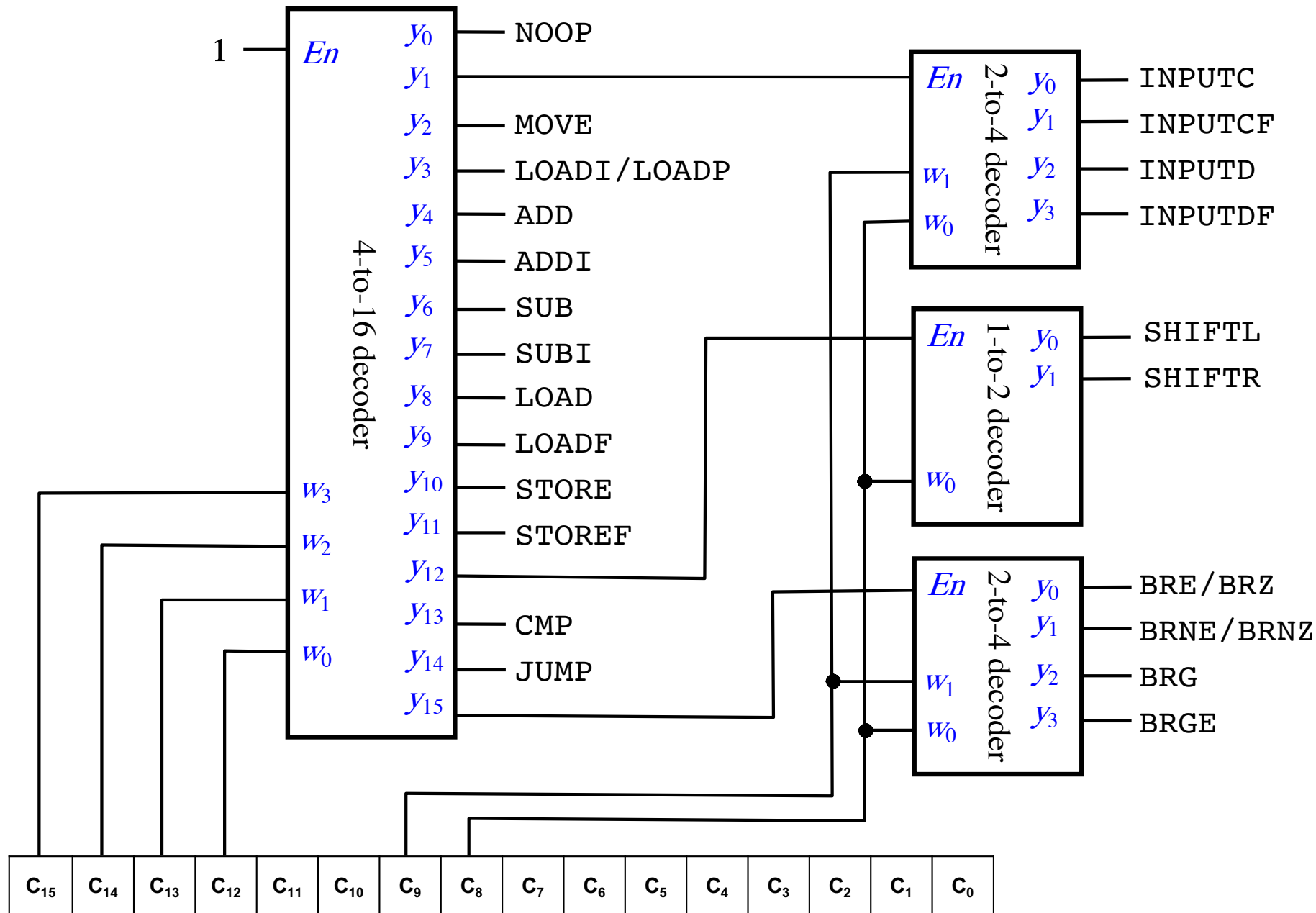


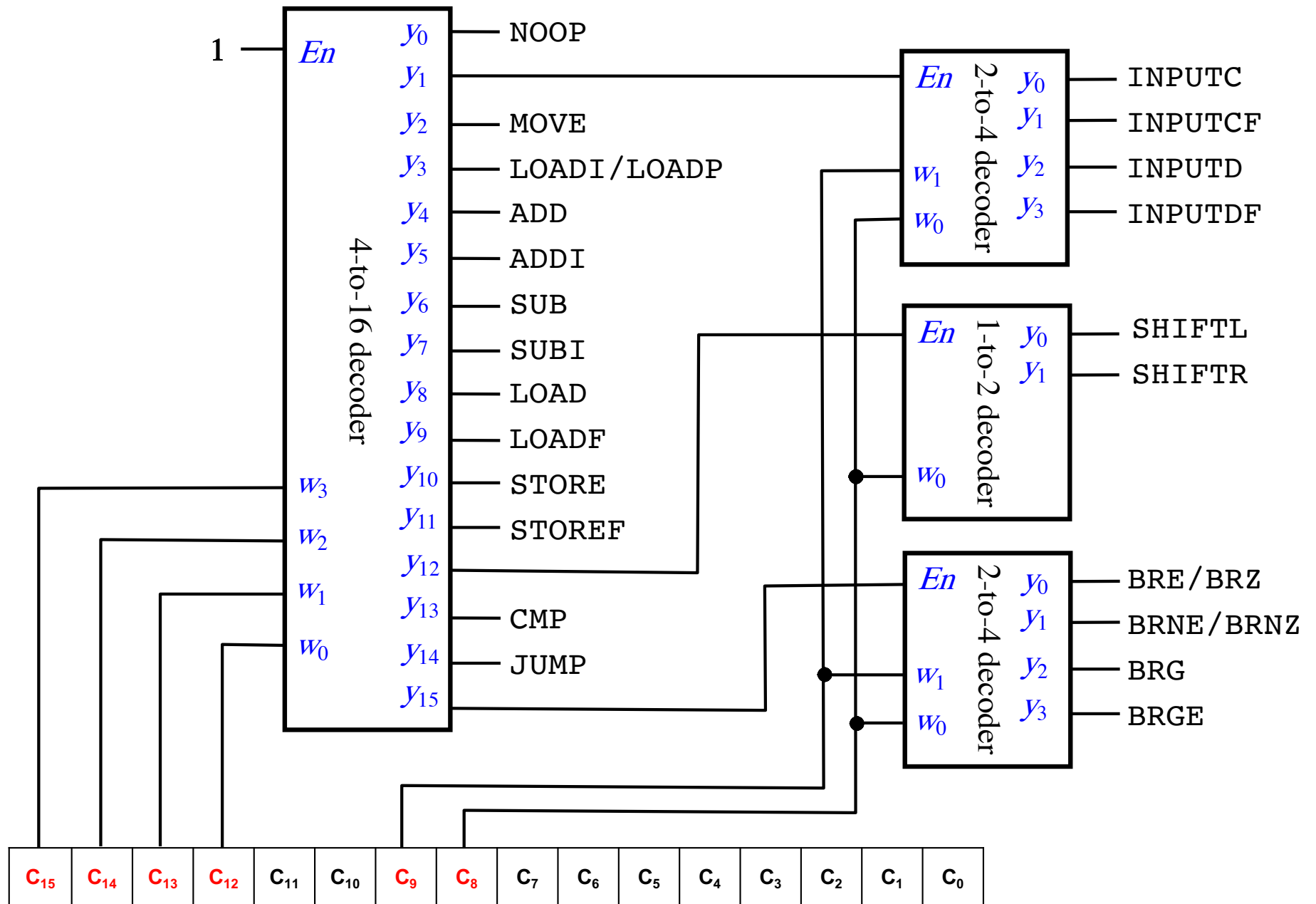




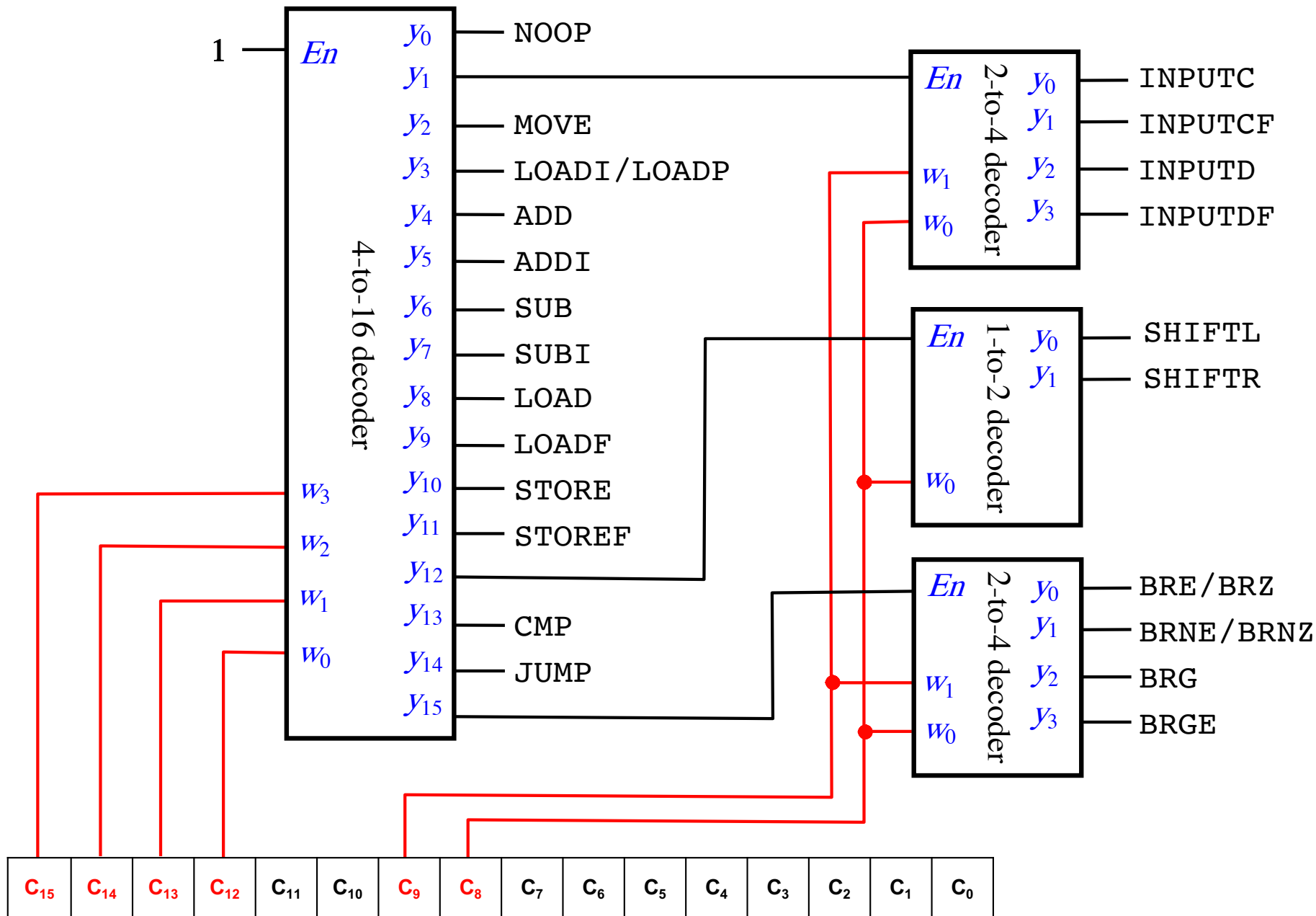




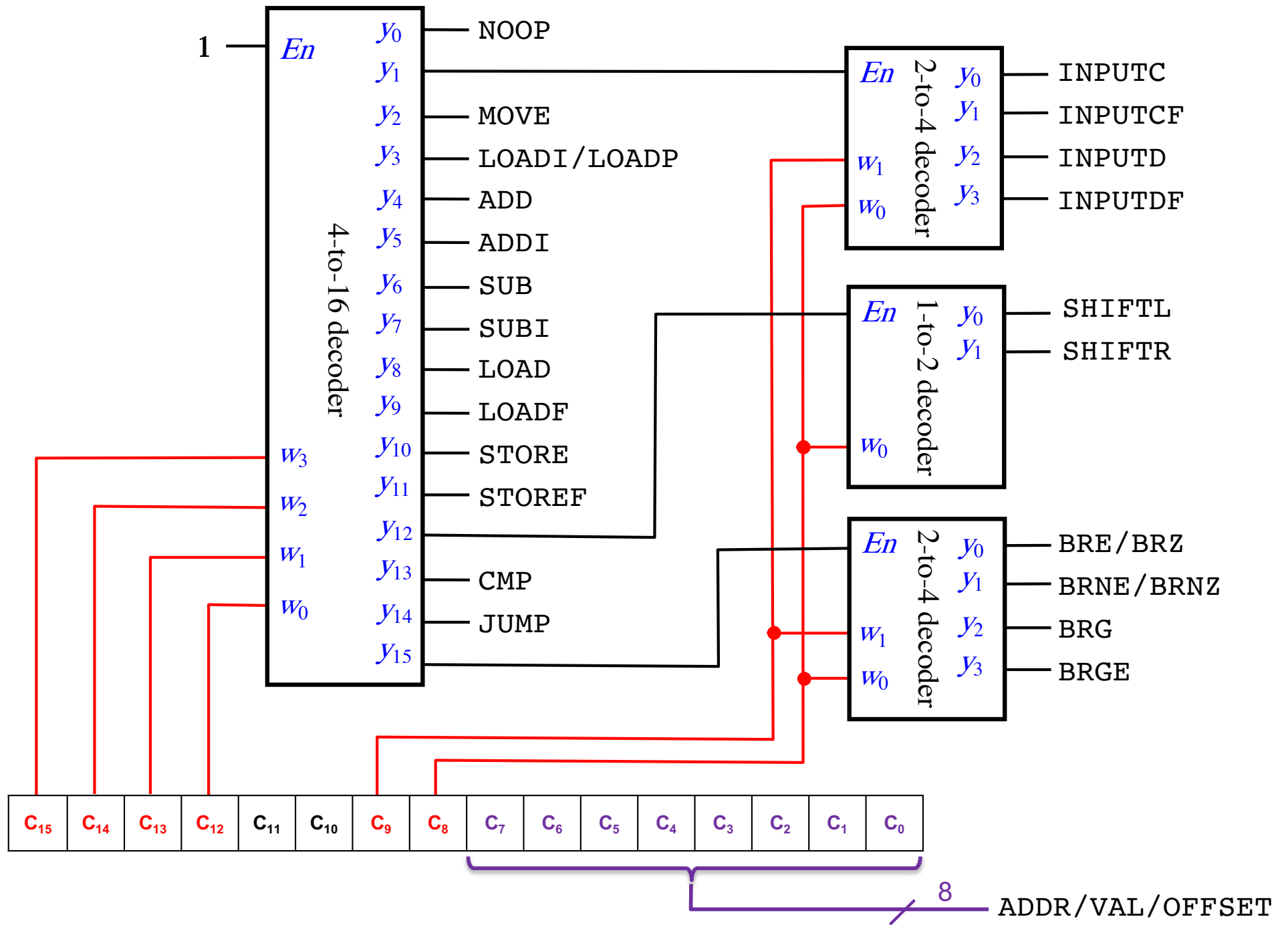


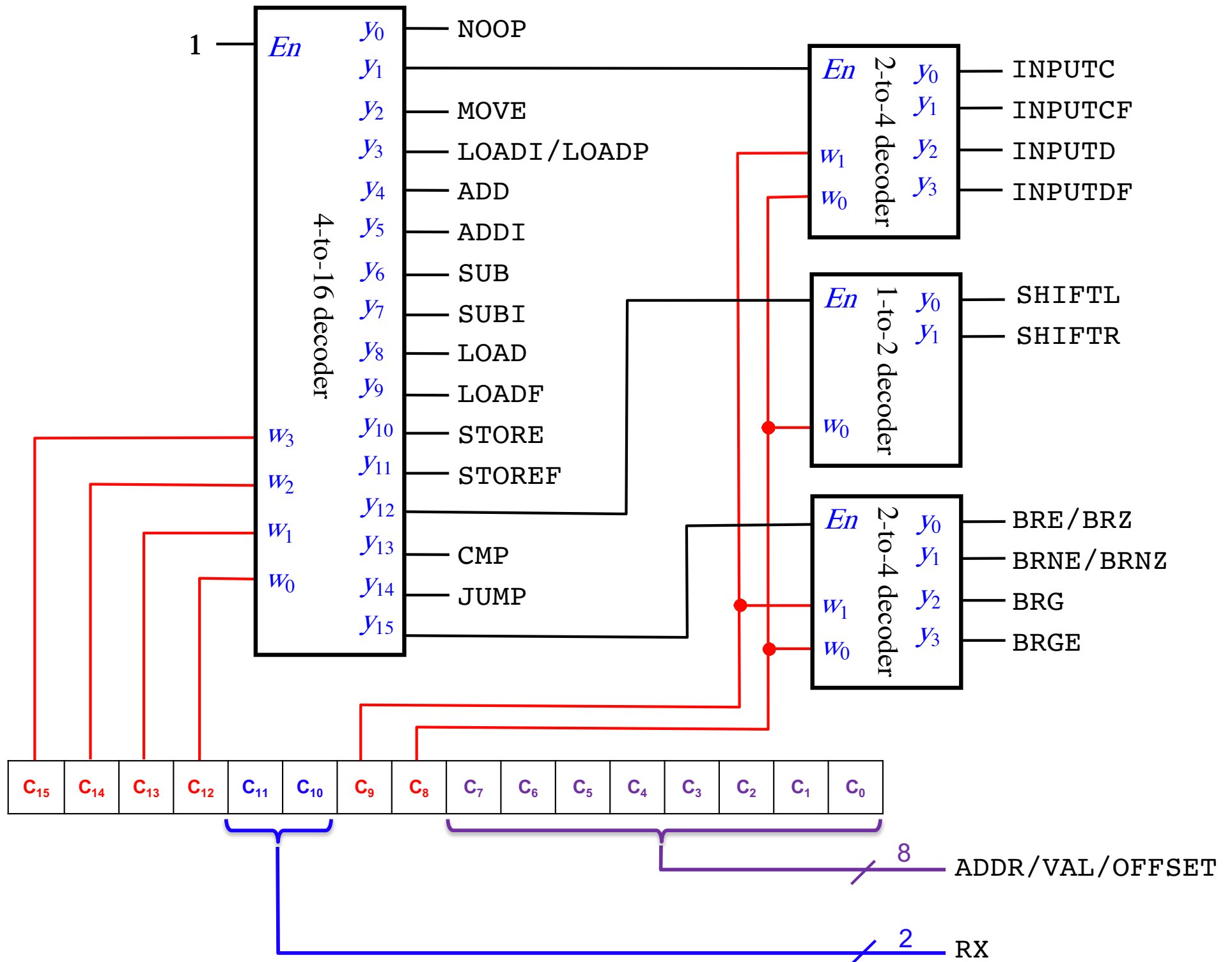


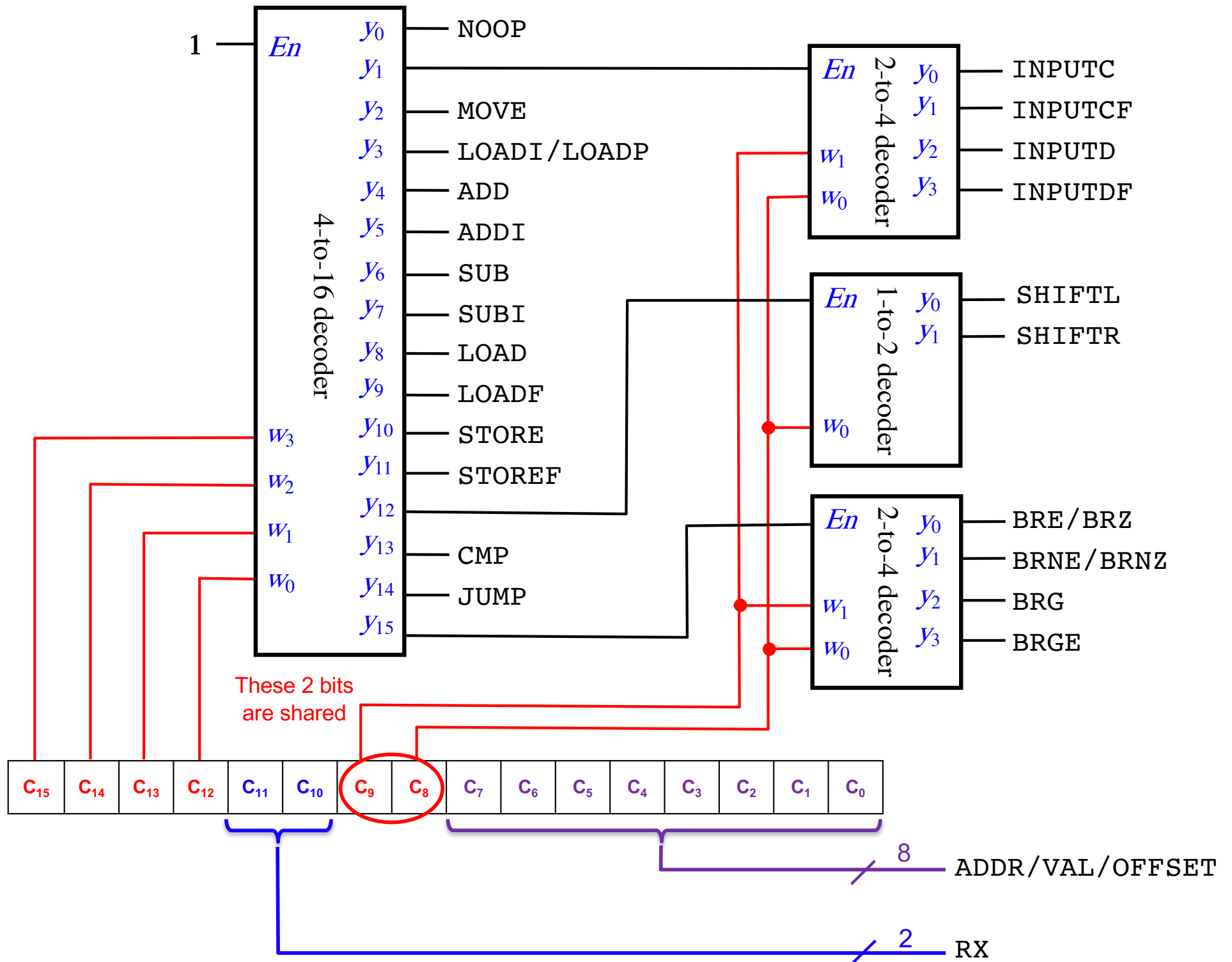
These 6 bits represent the OPCODEs

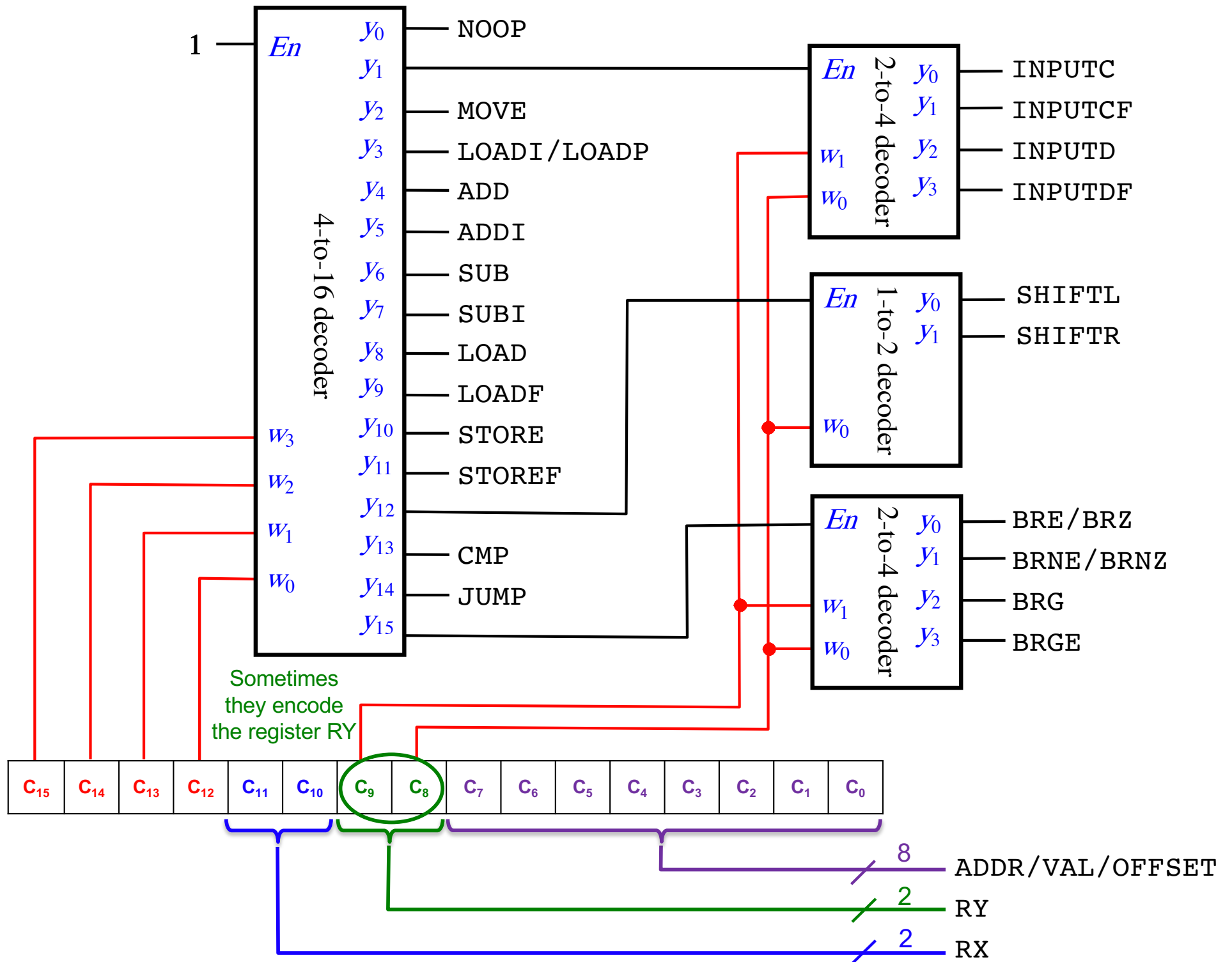


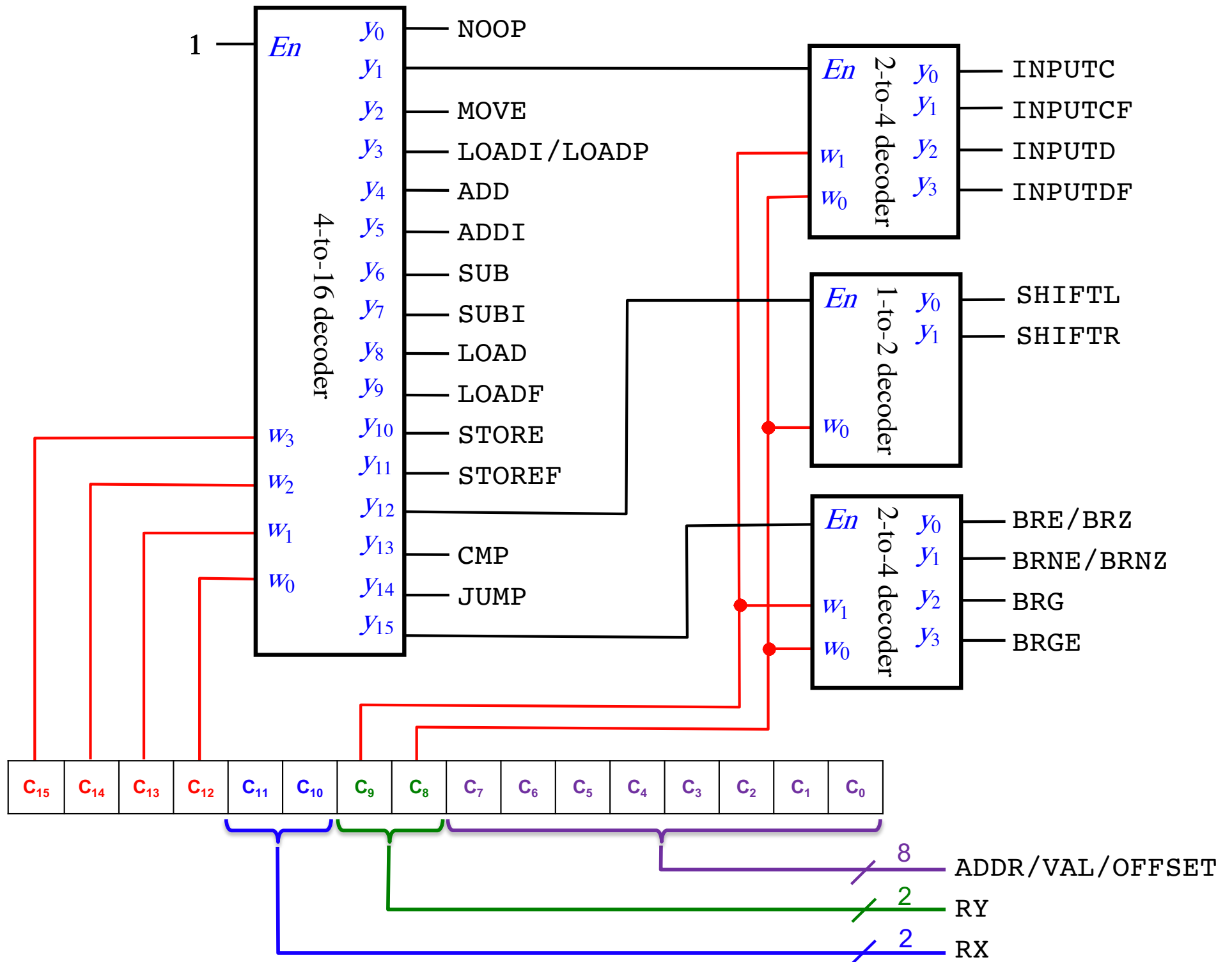
These 6 bits represent the OPCODEs

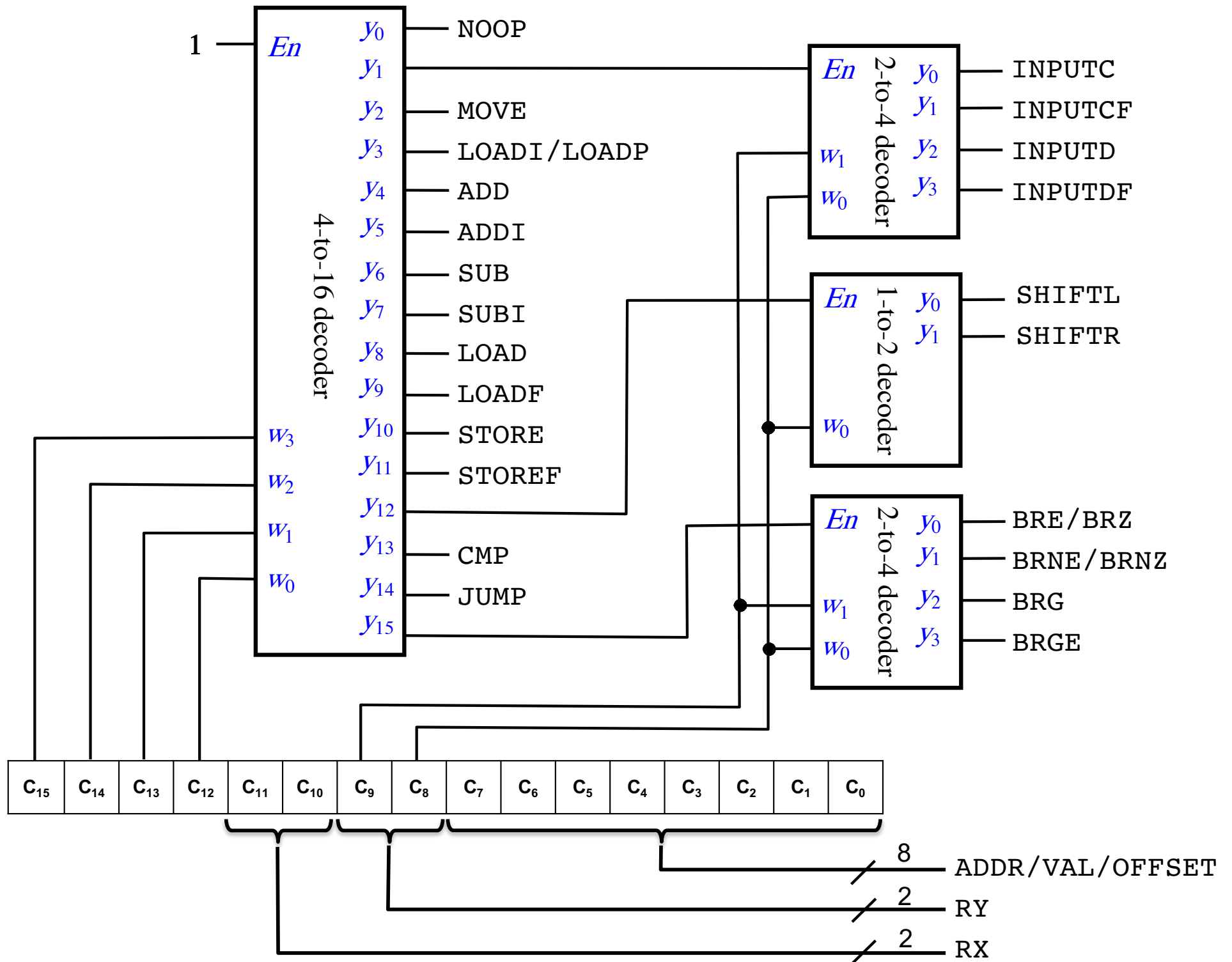




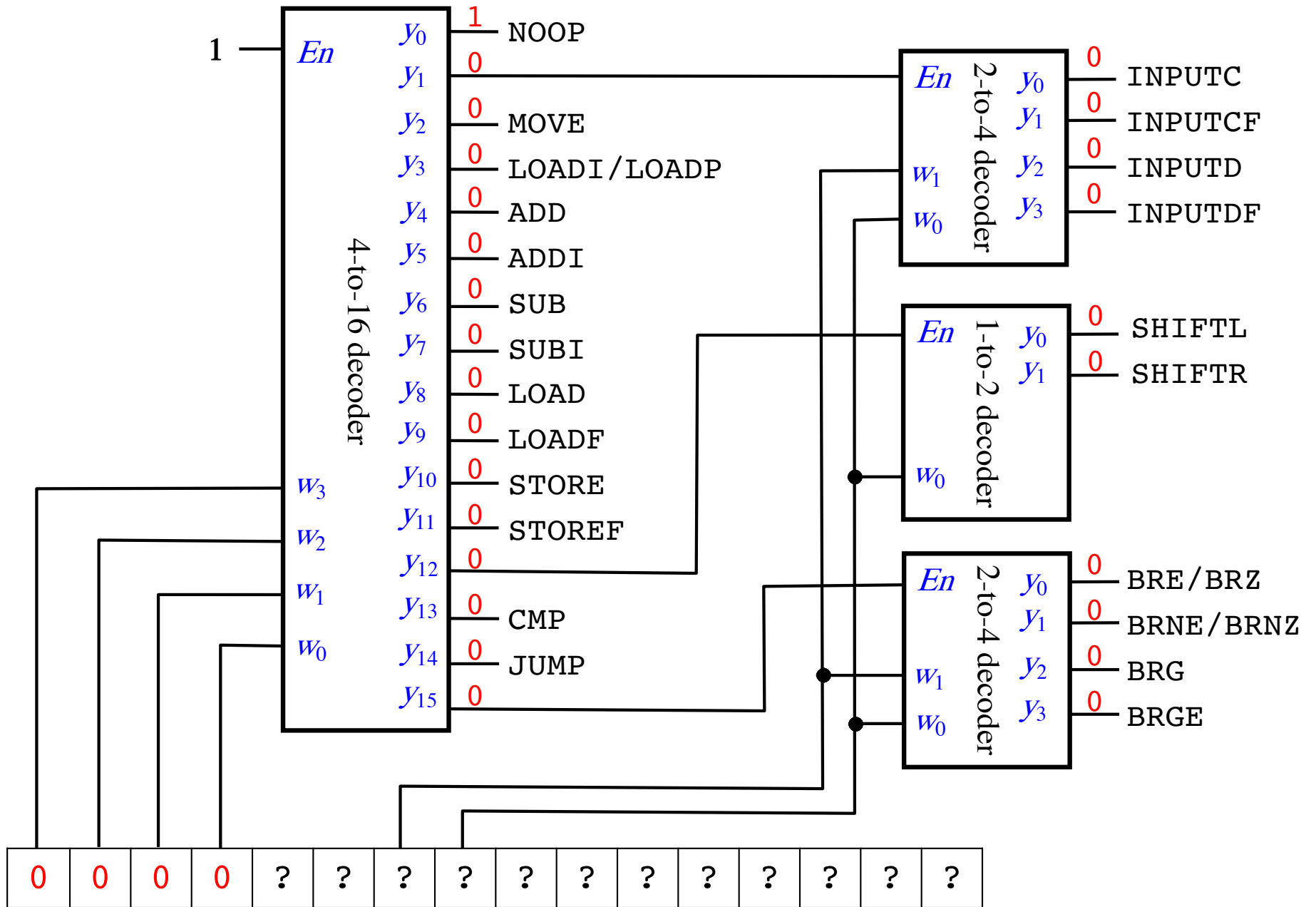




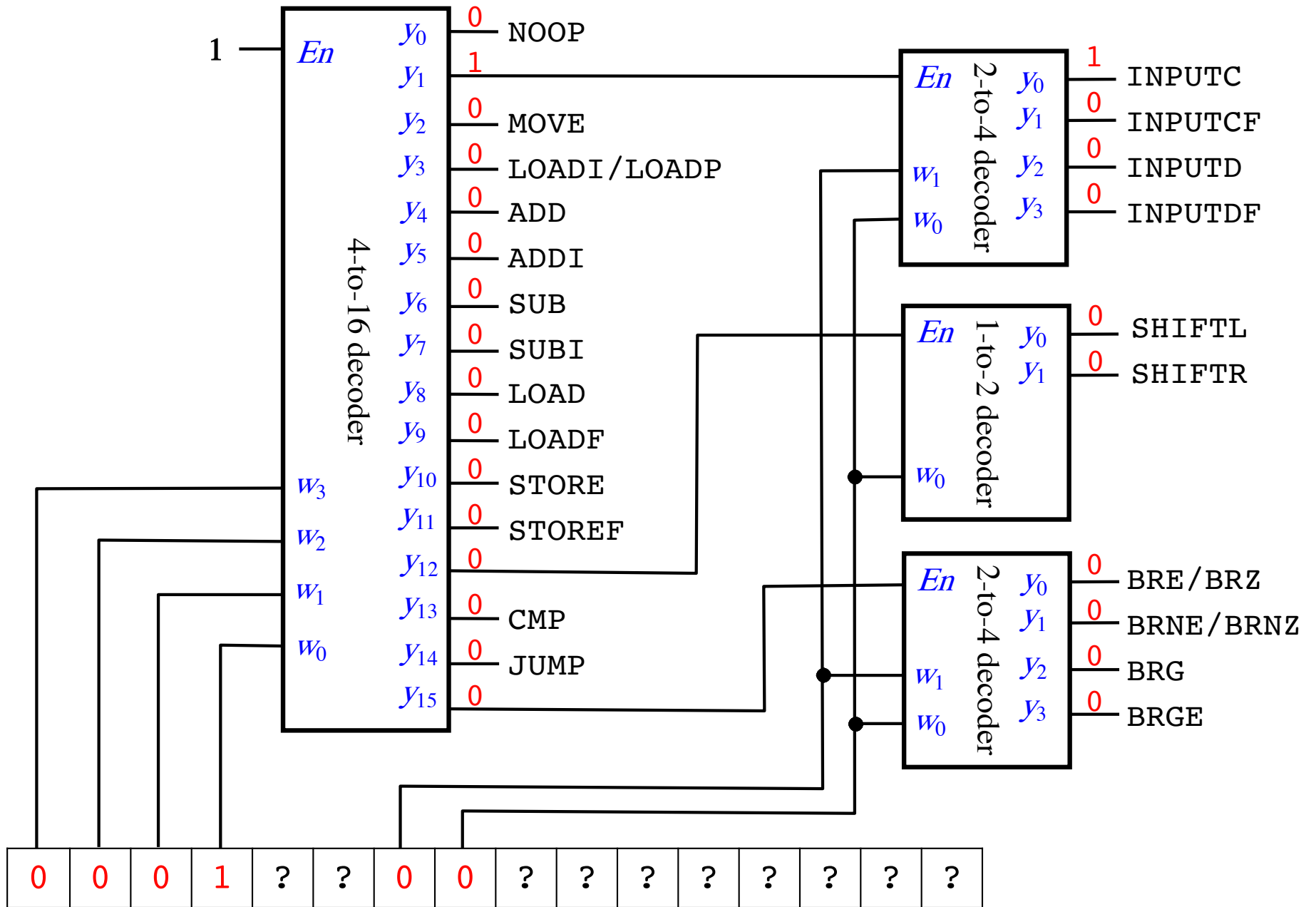




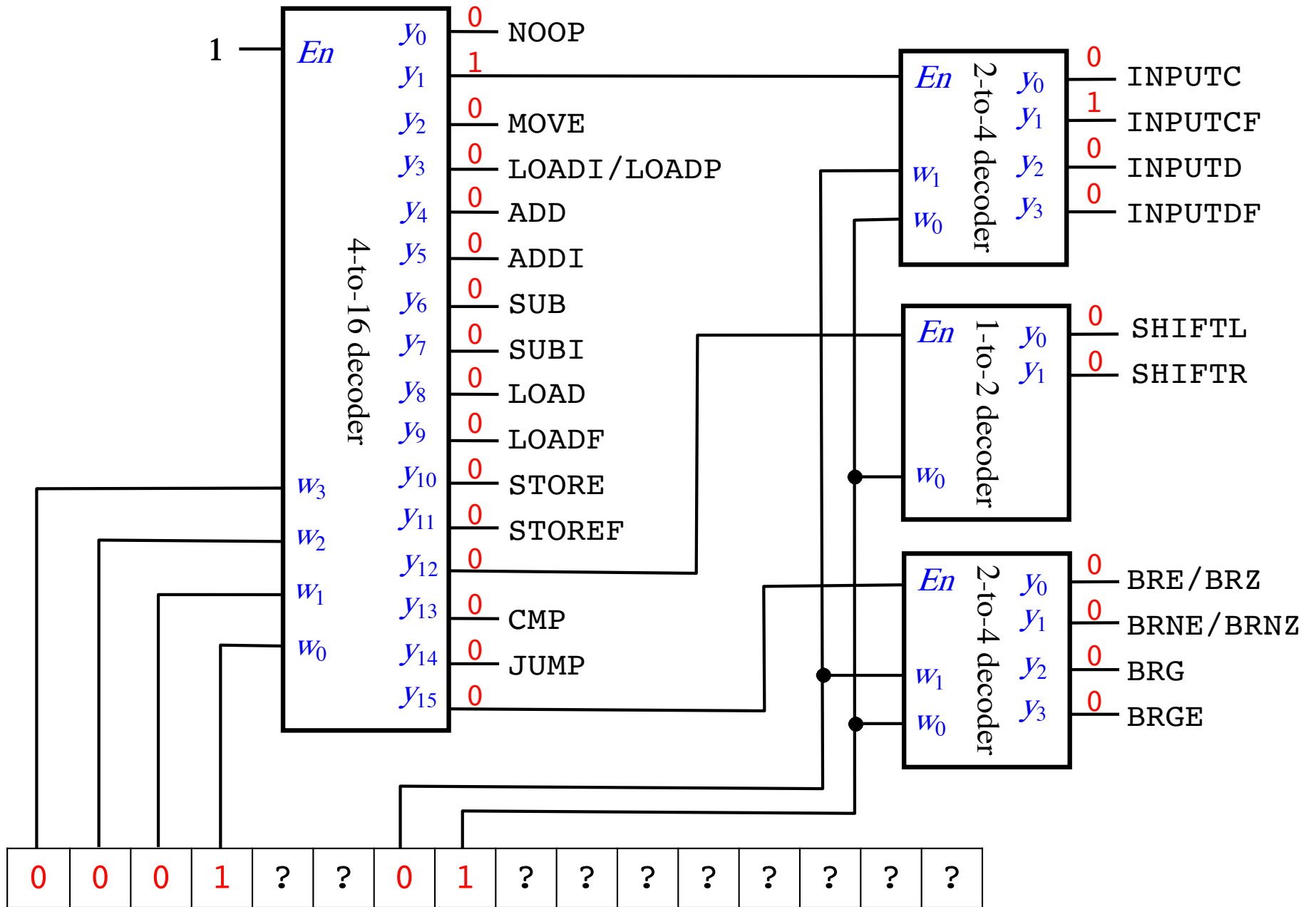
**The OPCODE decoder outputs
are one-hot encoded**



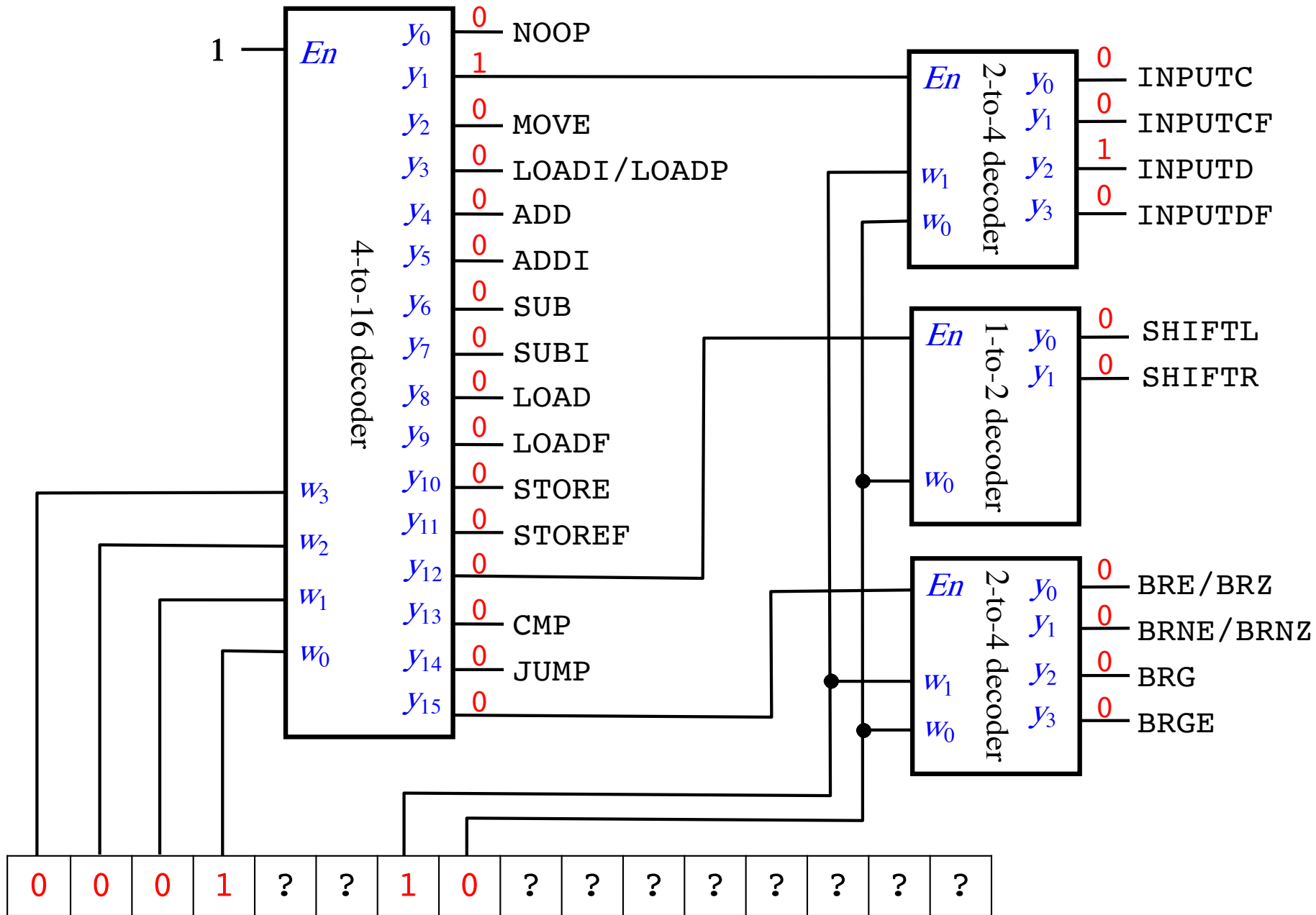
NOOP



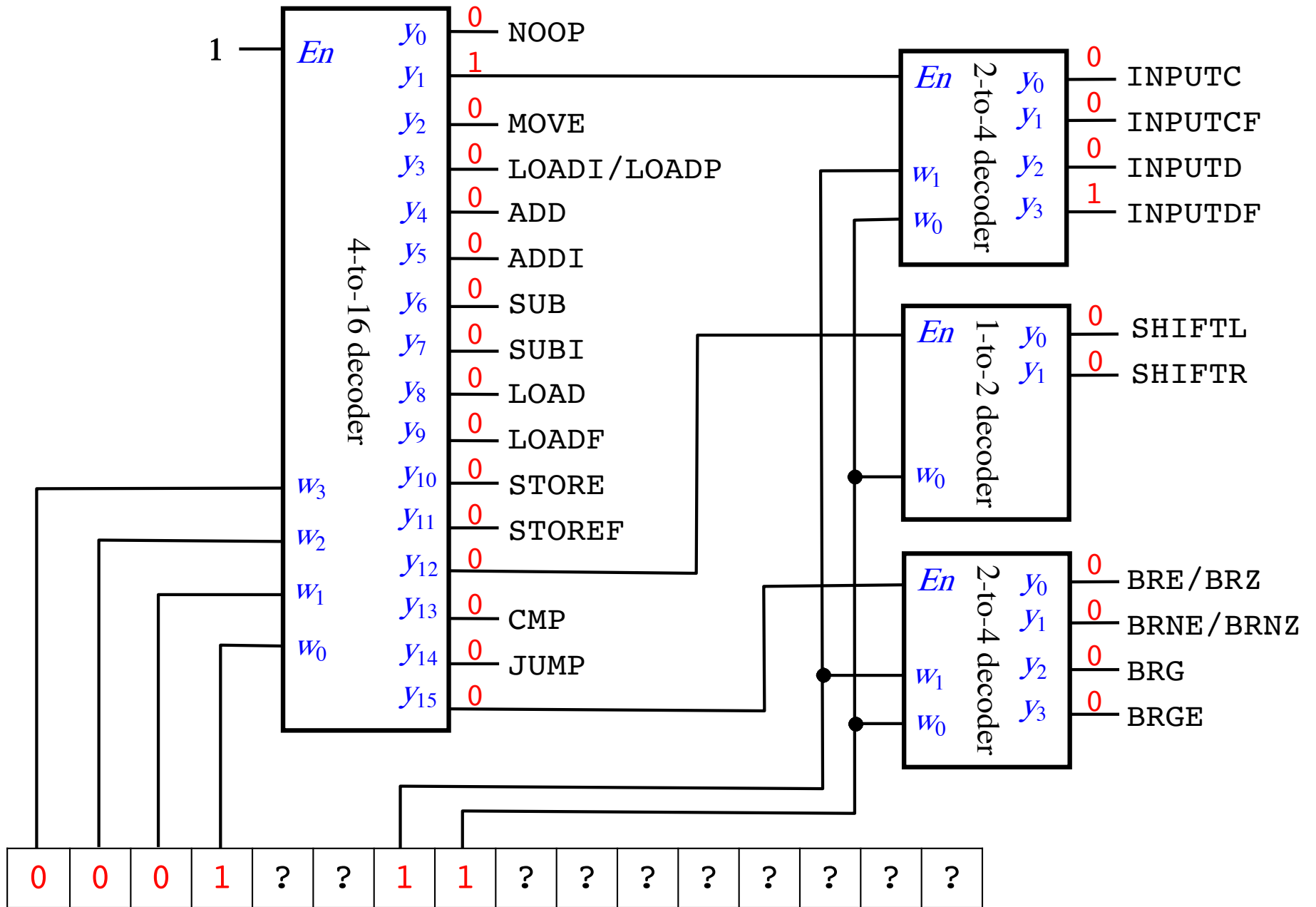
INPUTC



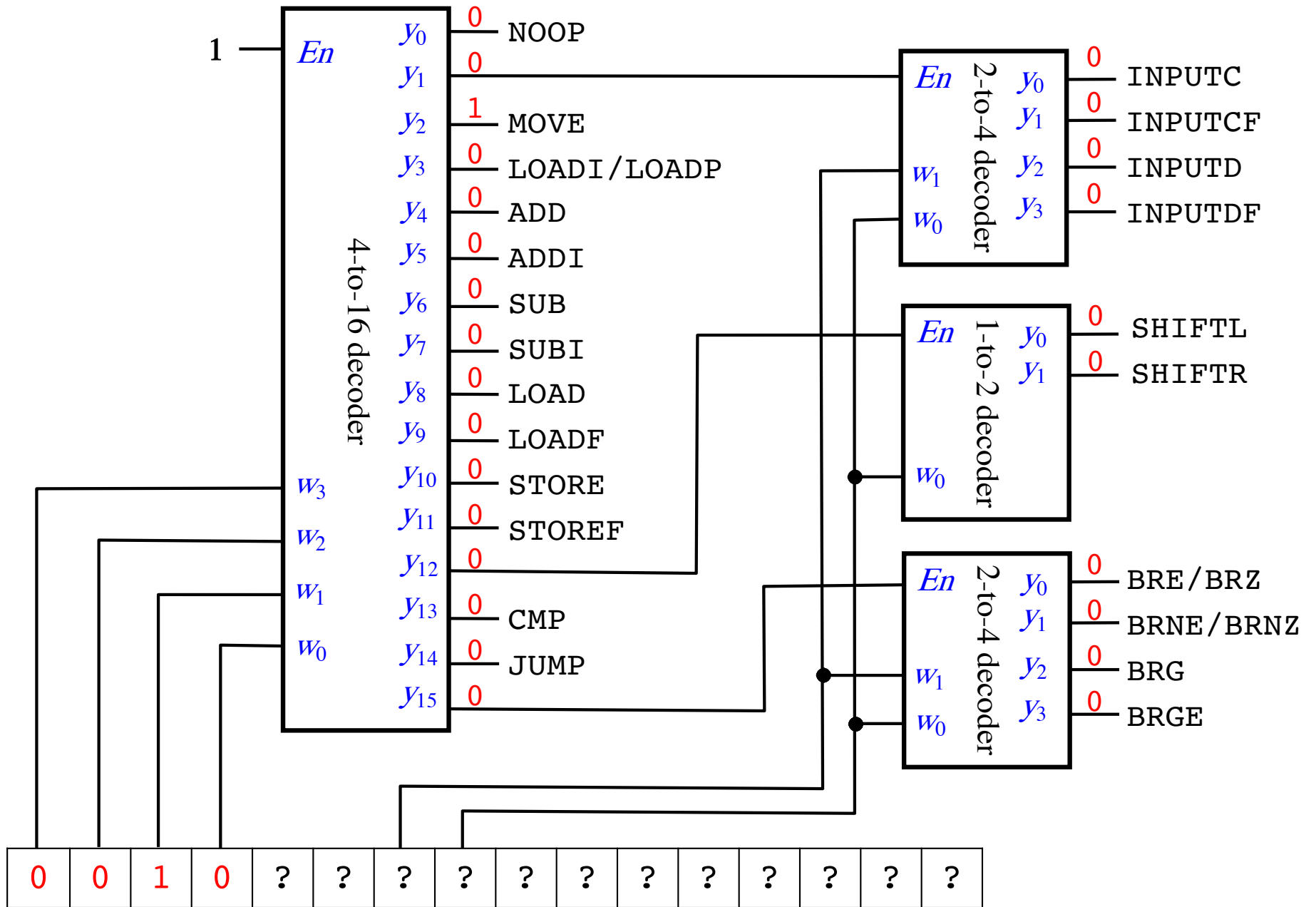
INPUTCF



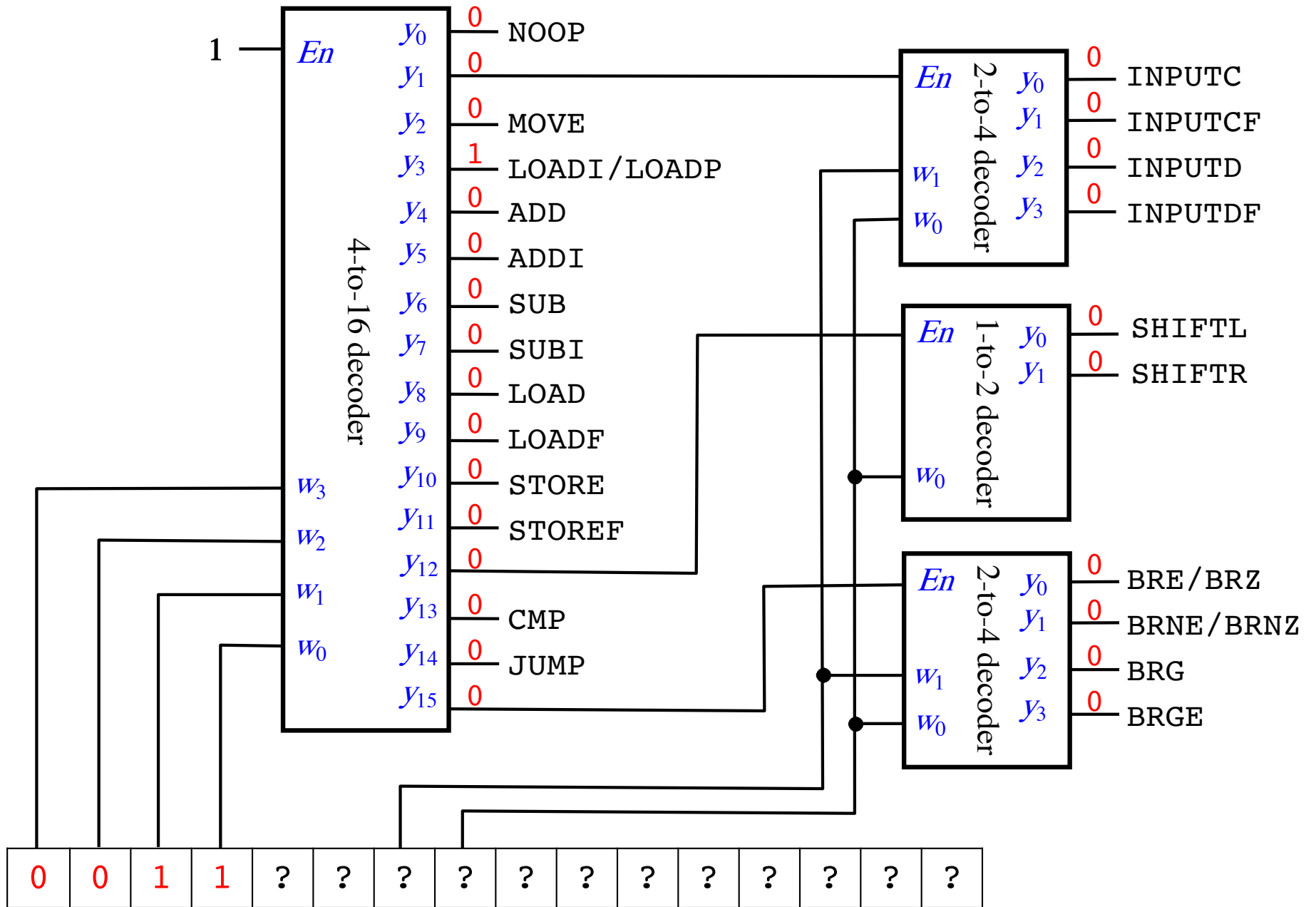
INPUTD



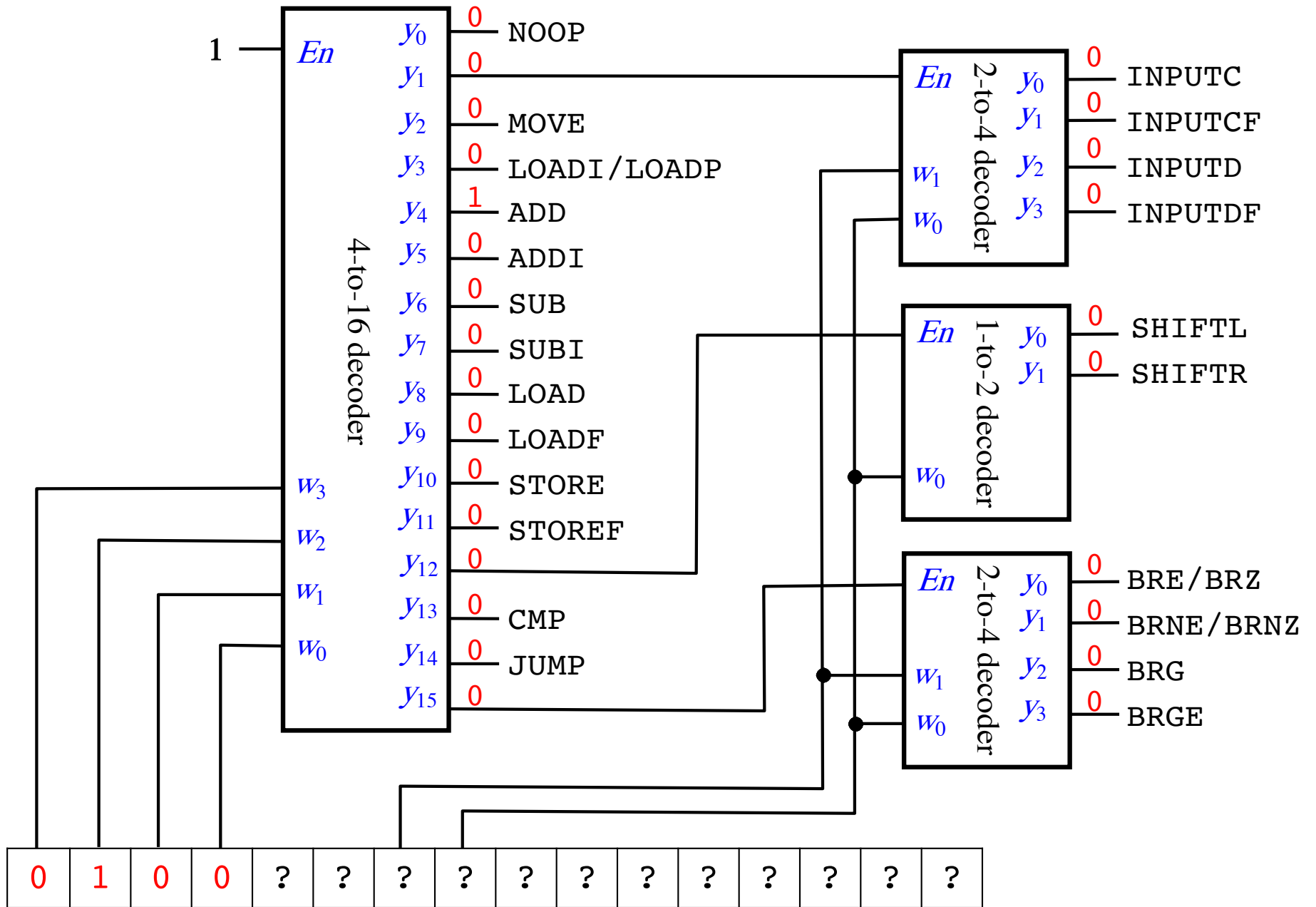
INPUTDF



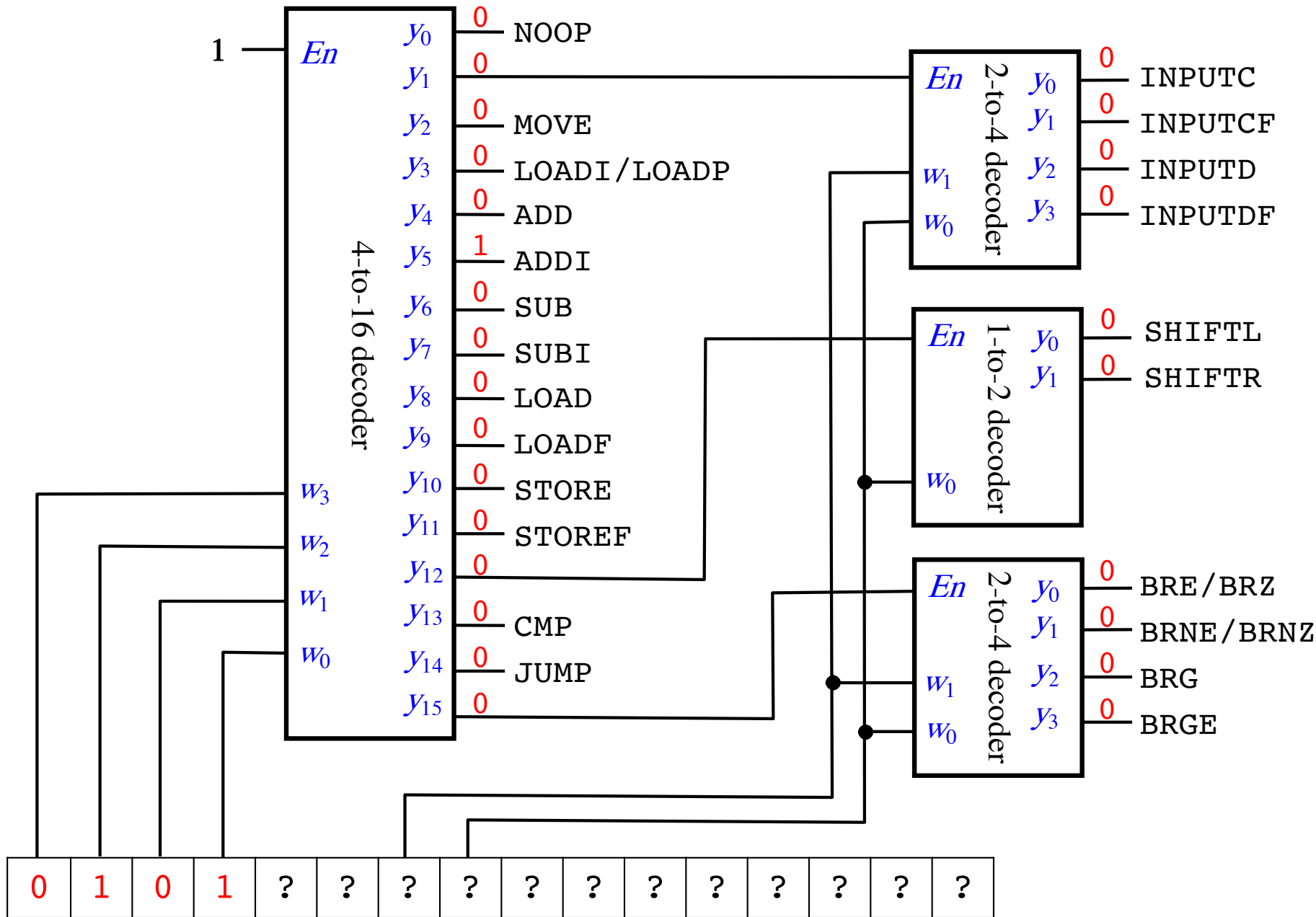
MOVE



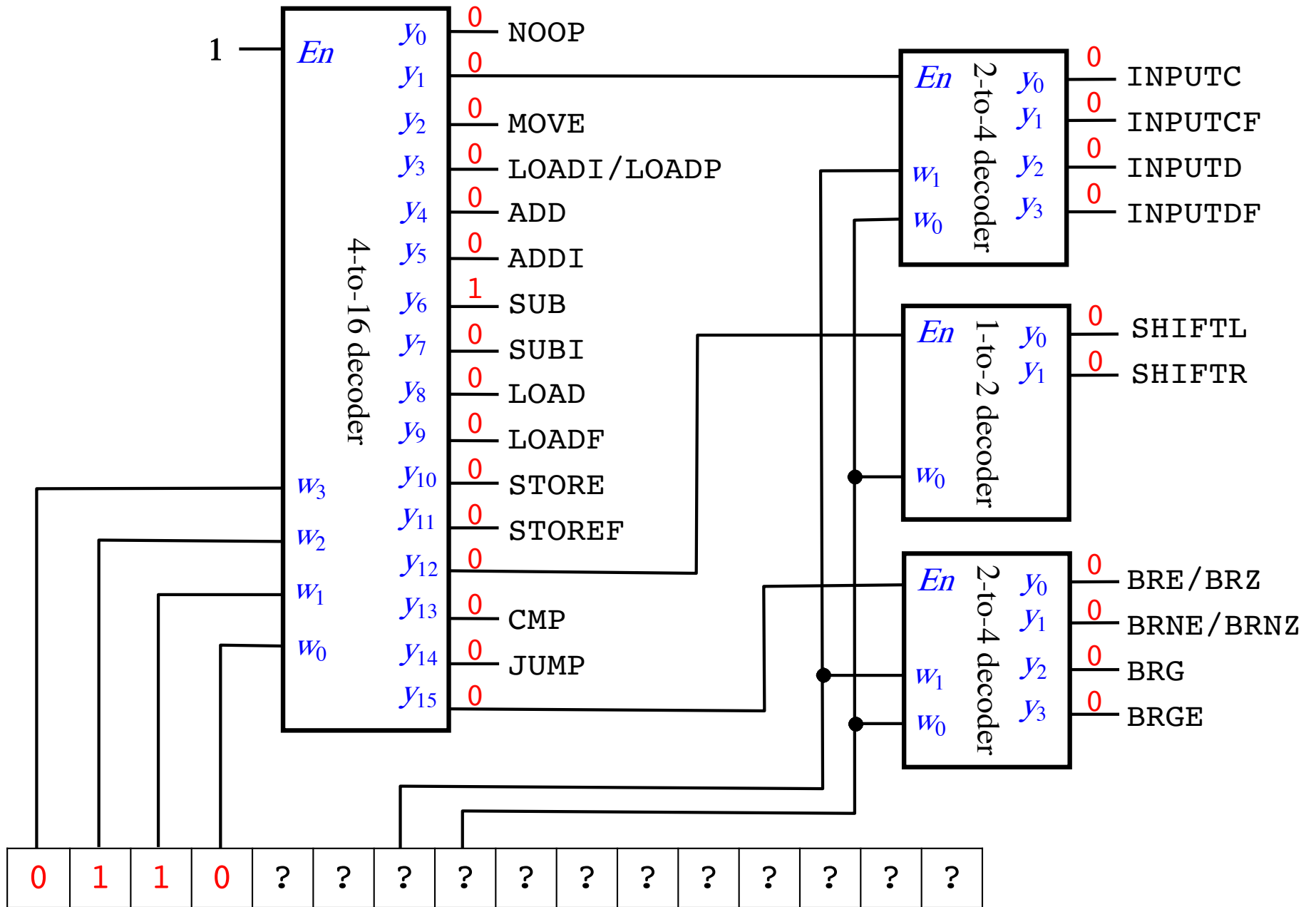
LOADI / LOADP



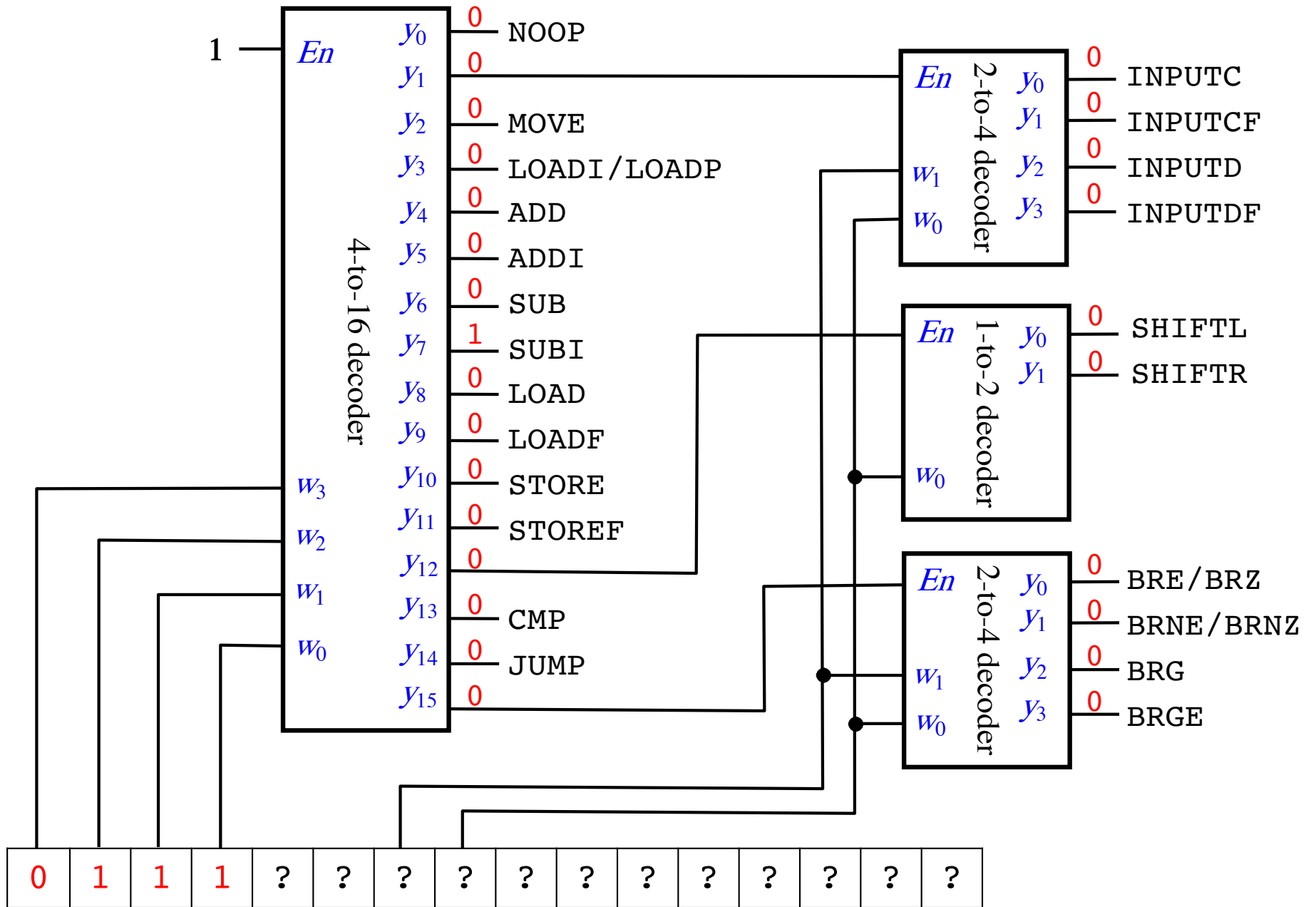
ADD



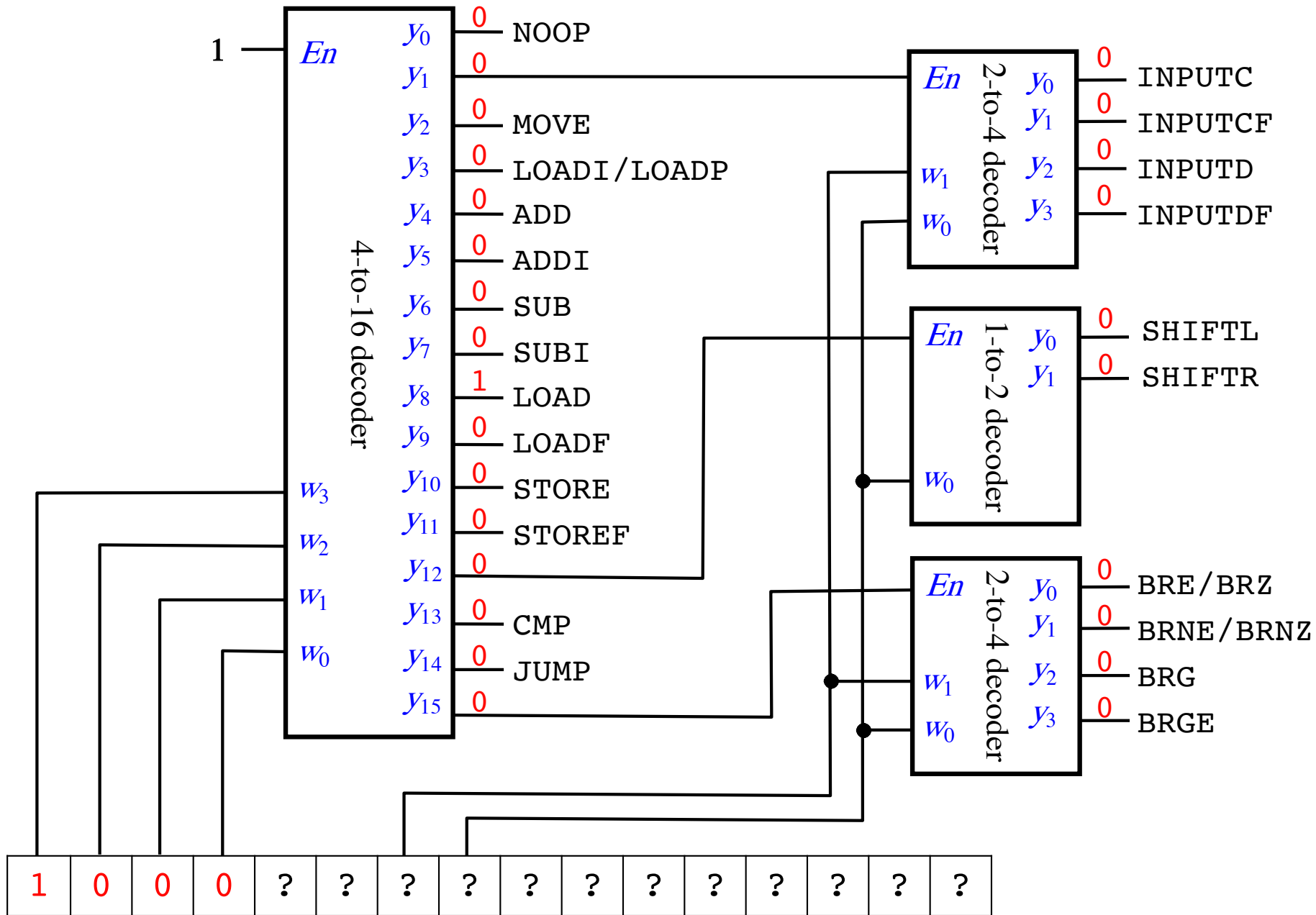
ADDI



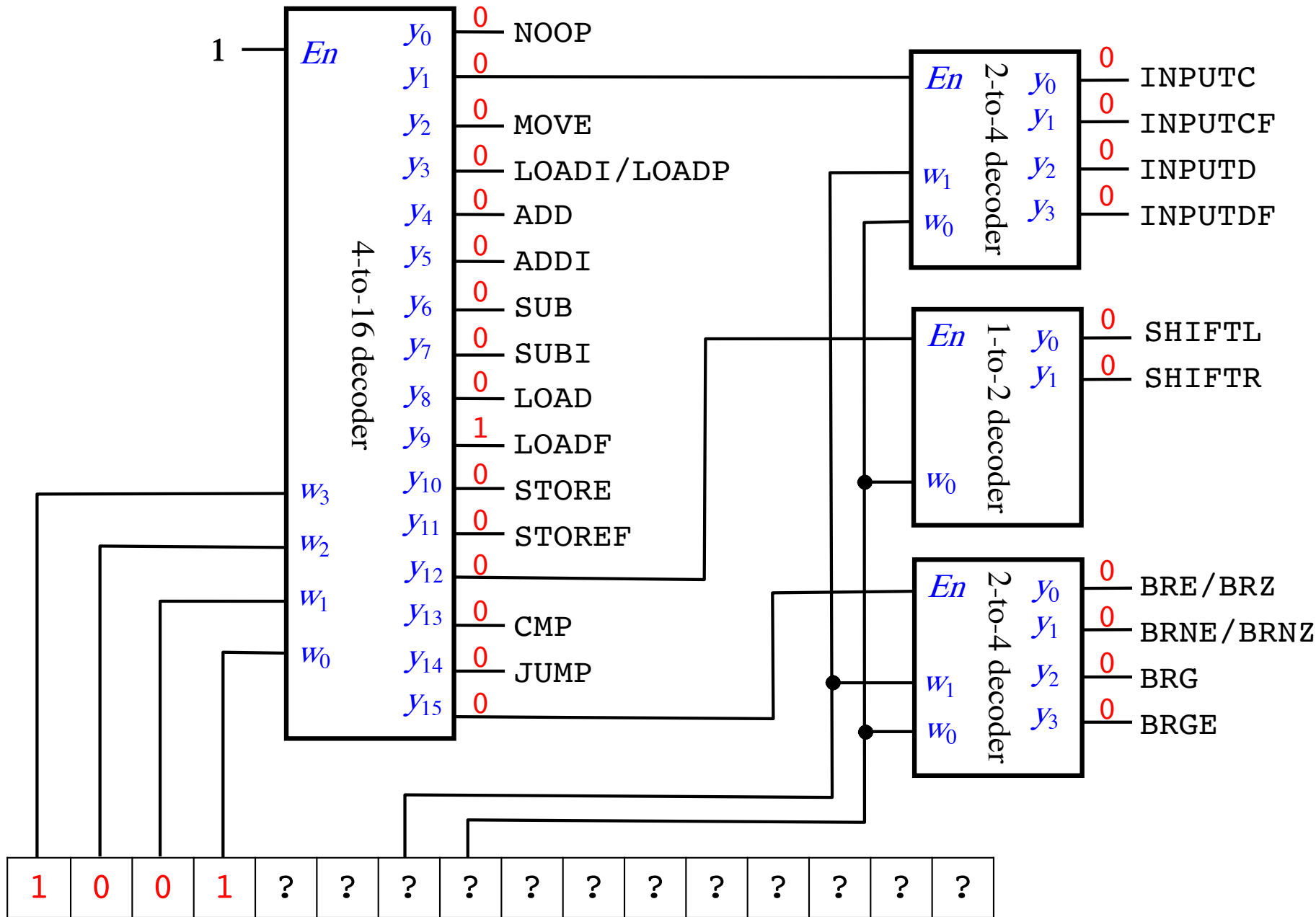
SUB



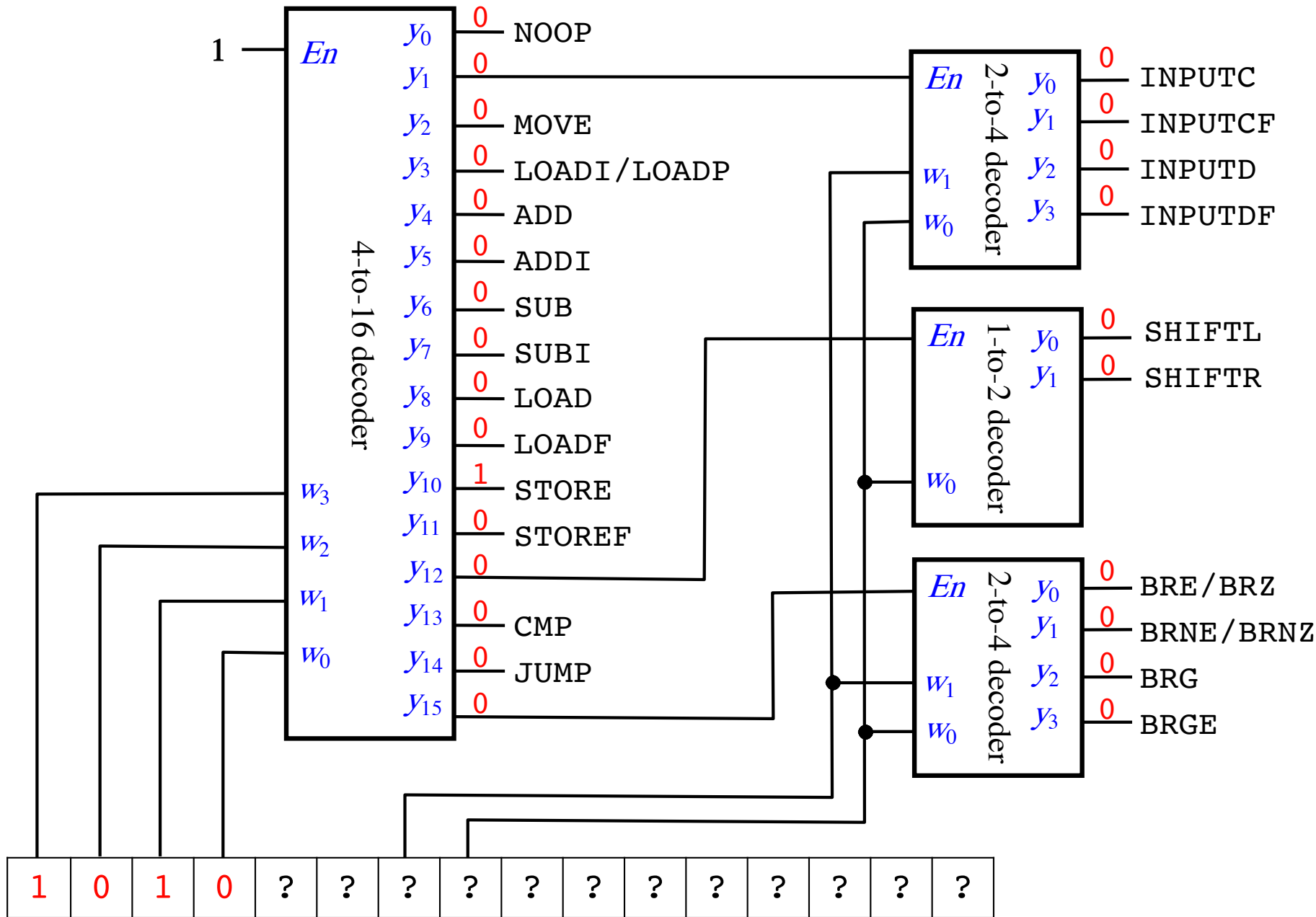
SUBI



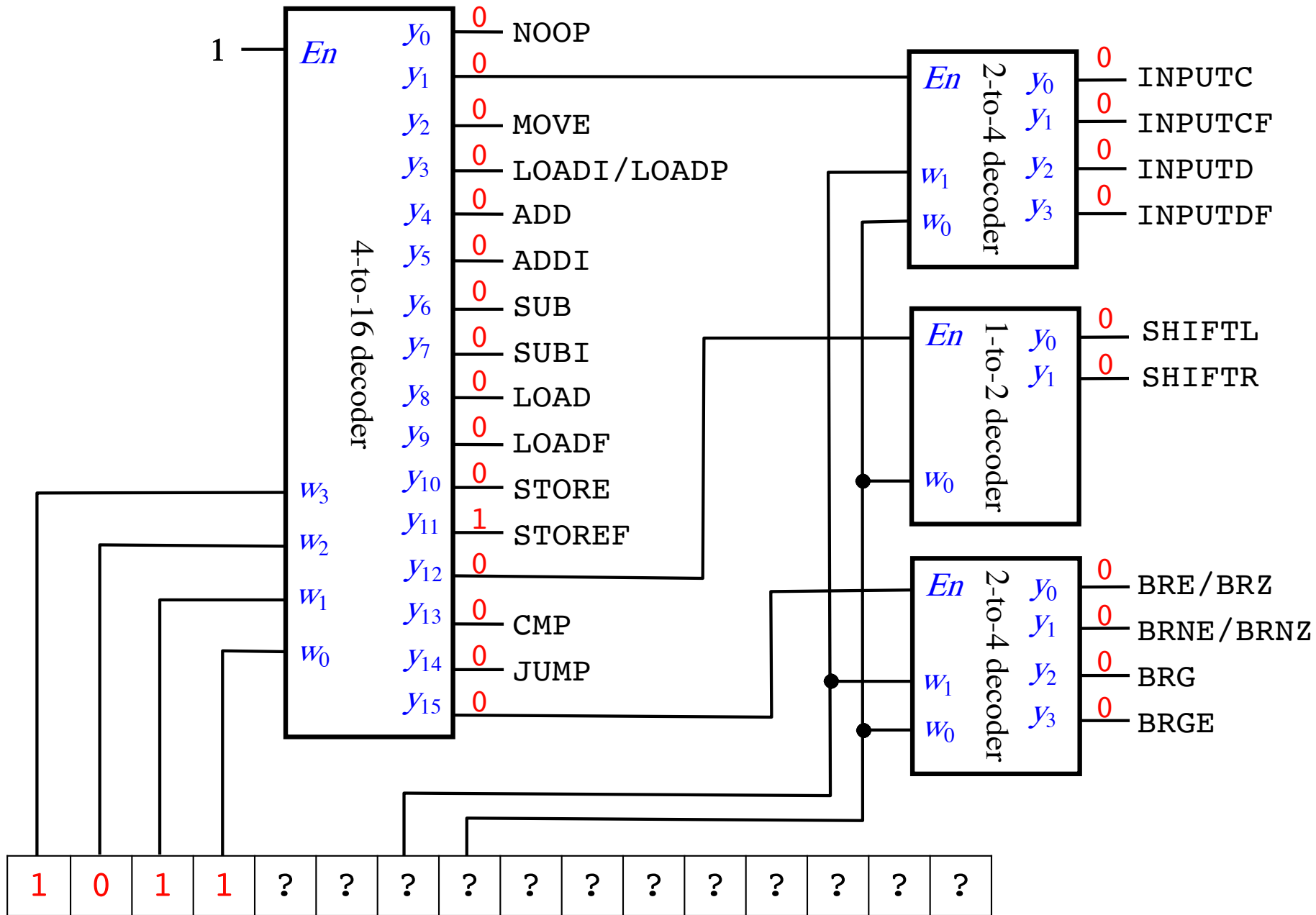
LOAD



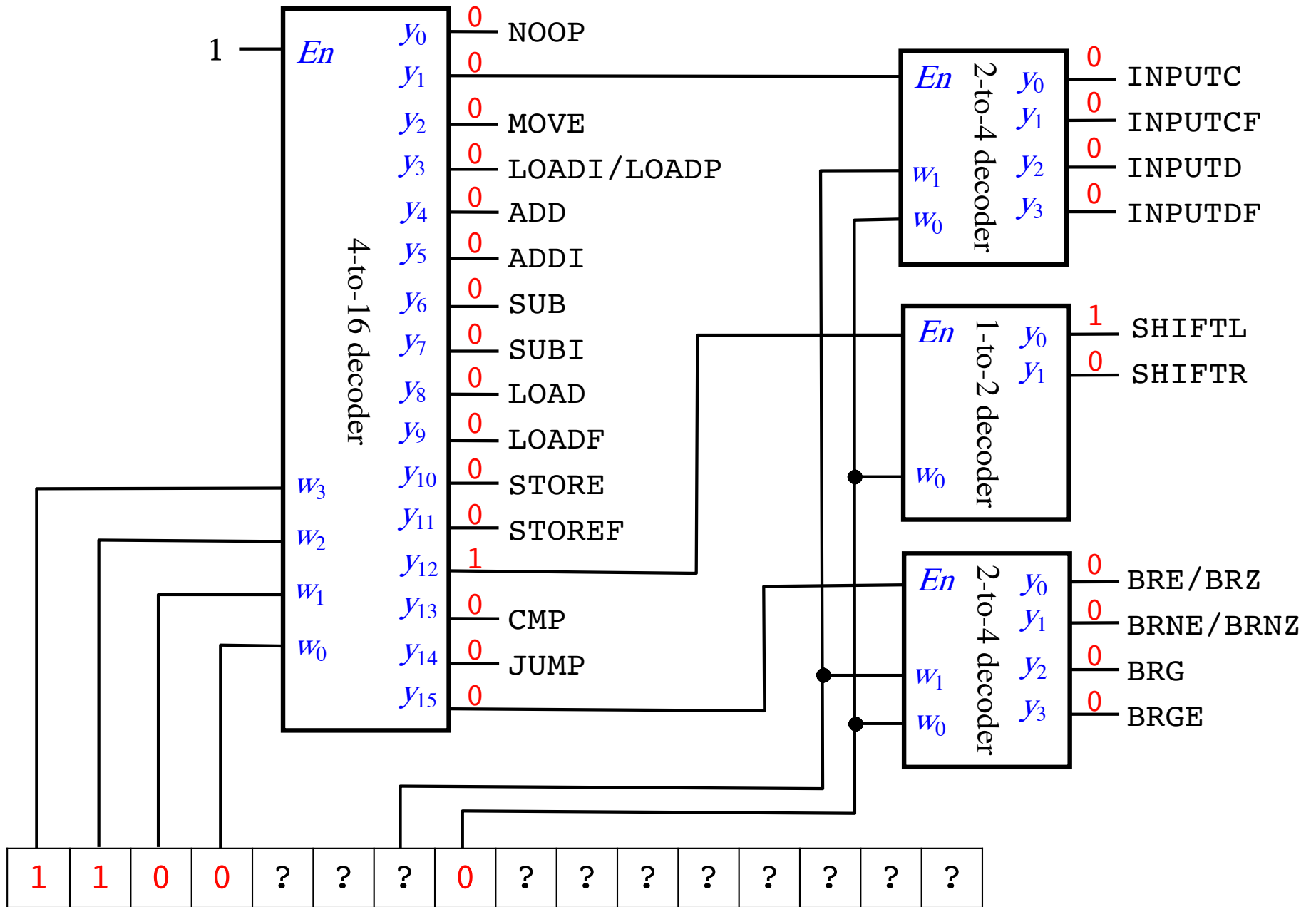
LOADF



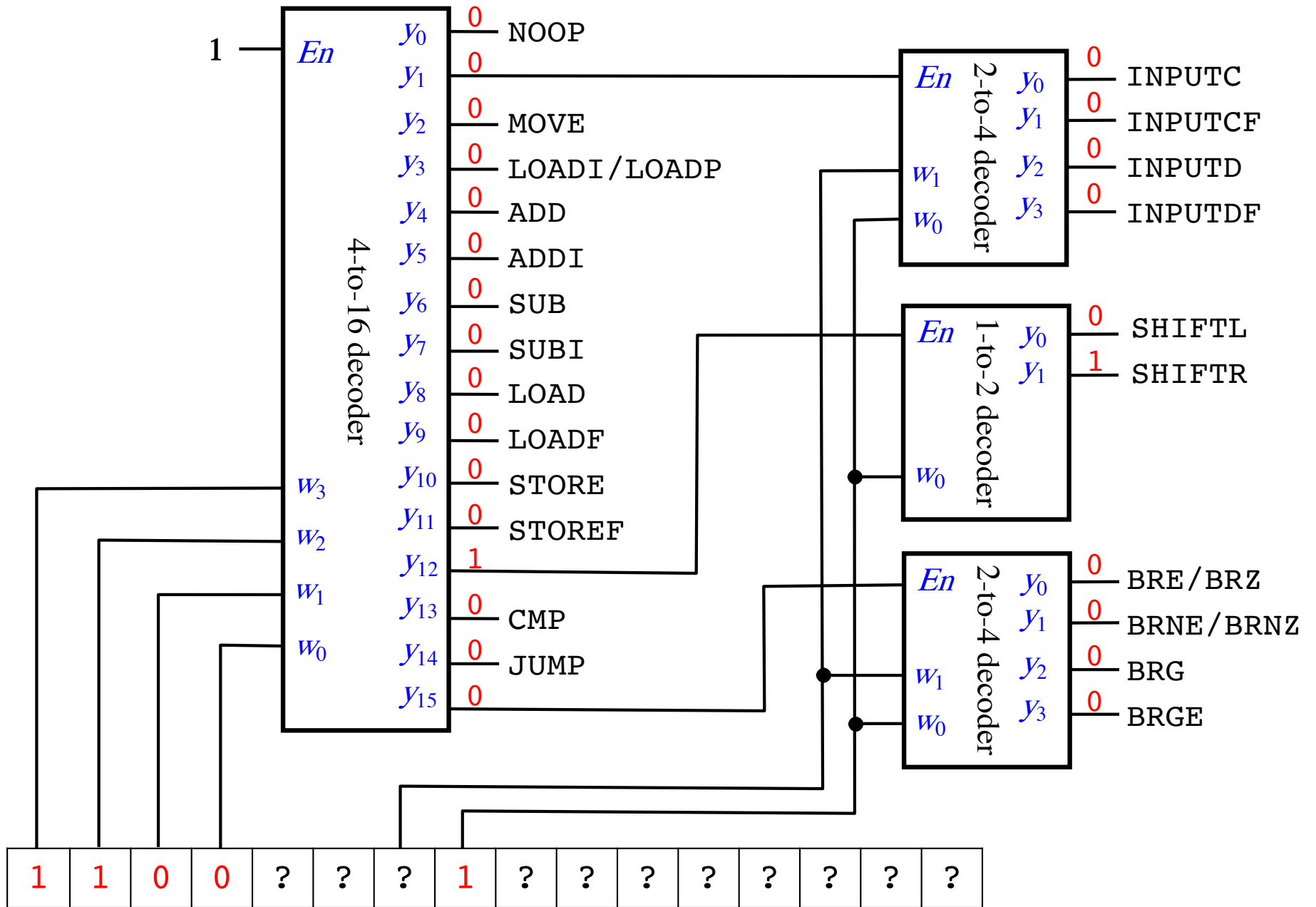
STORE



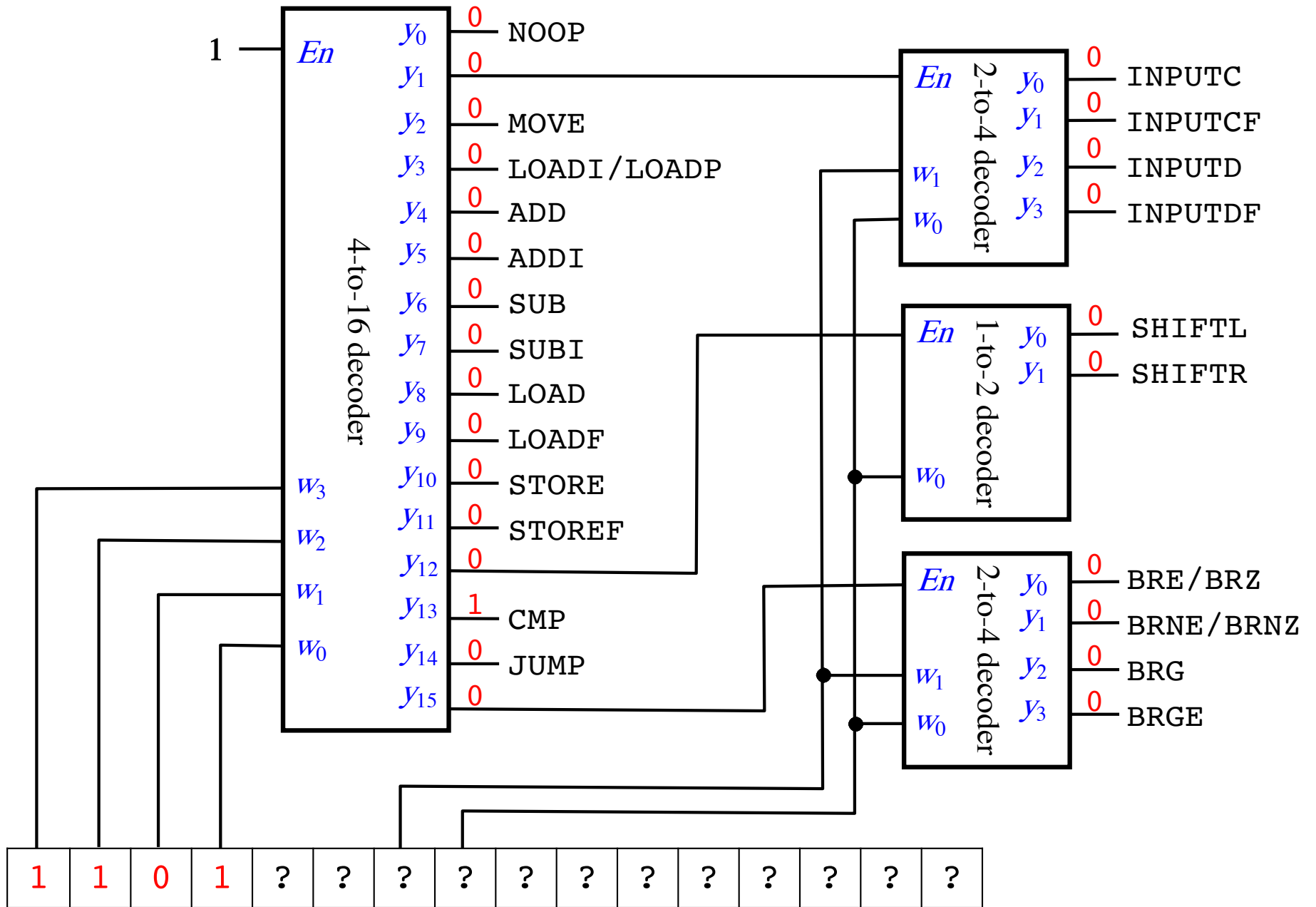
STOREF



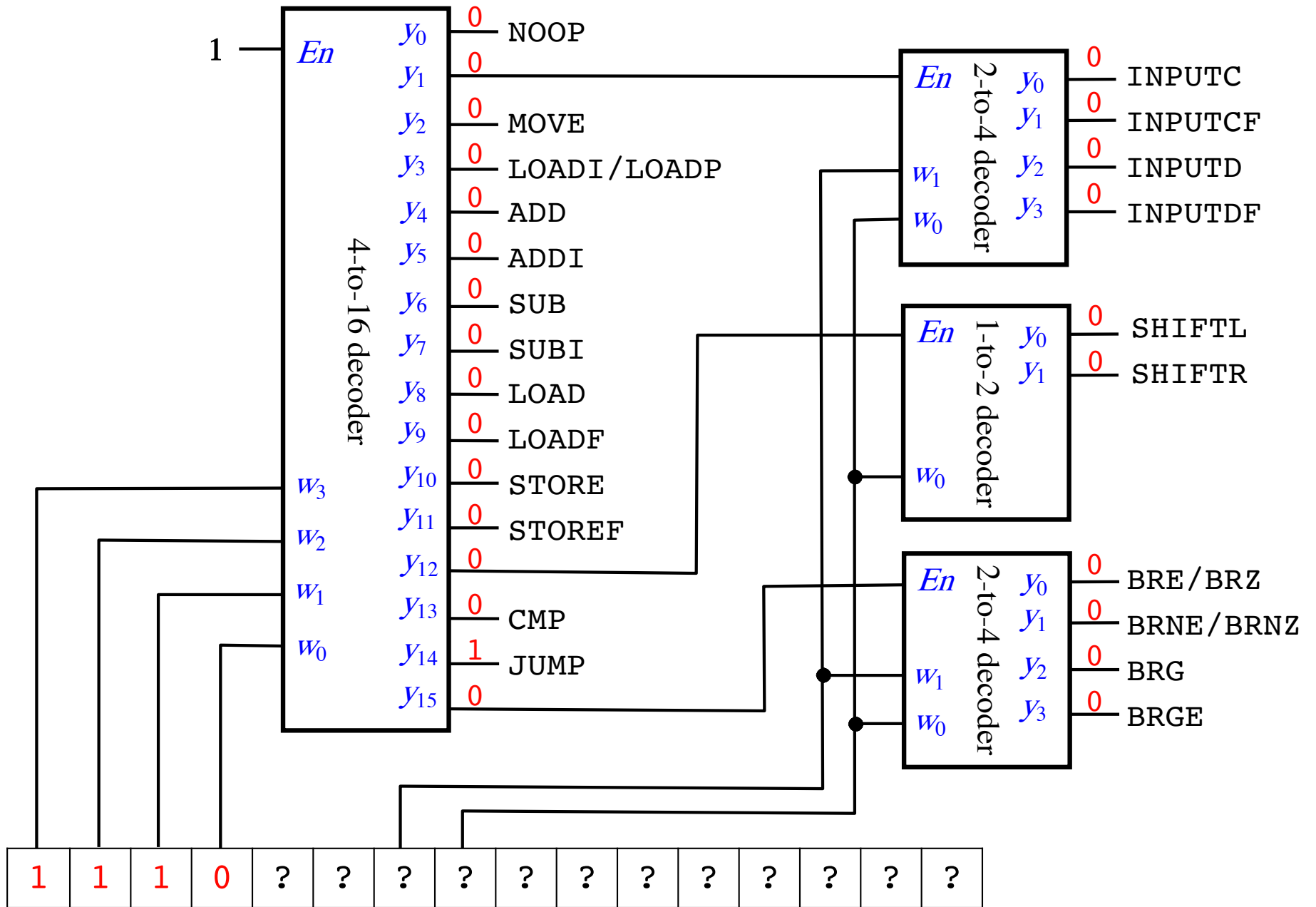
SHIFTL



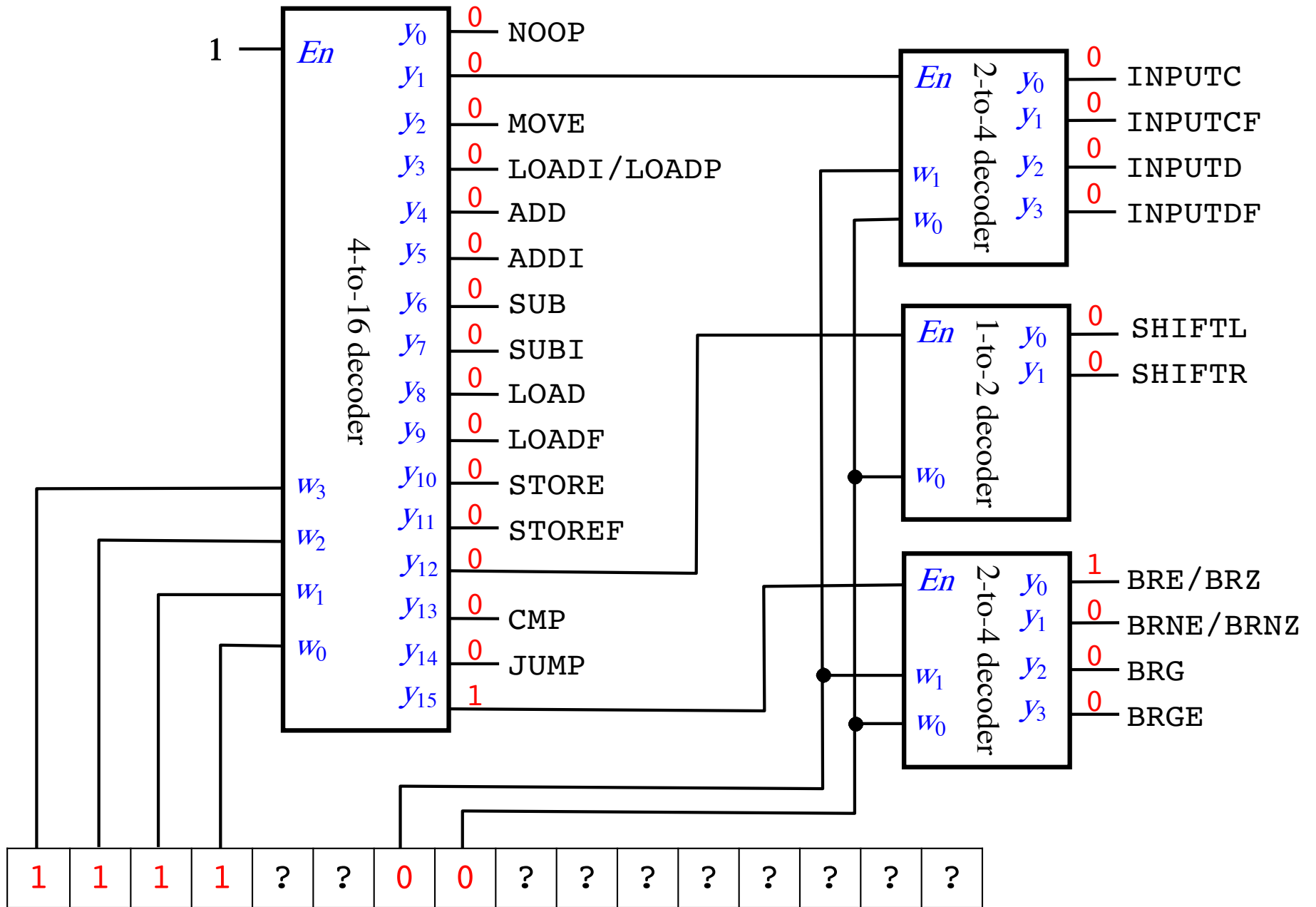
SHIFTR



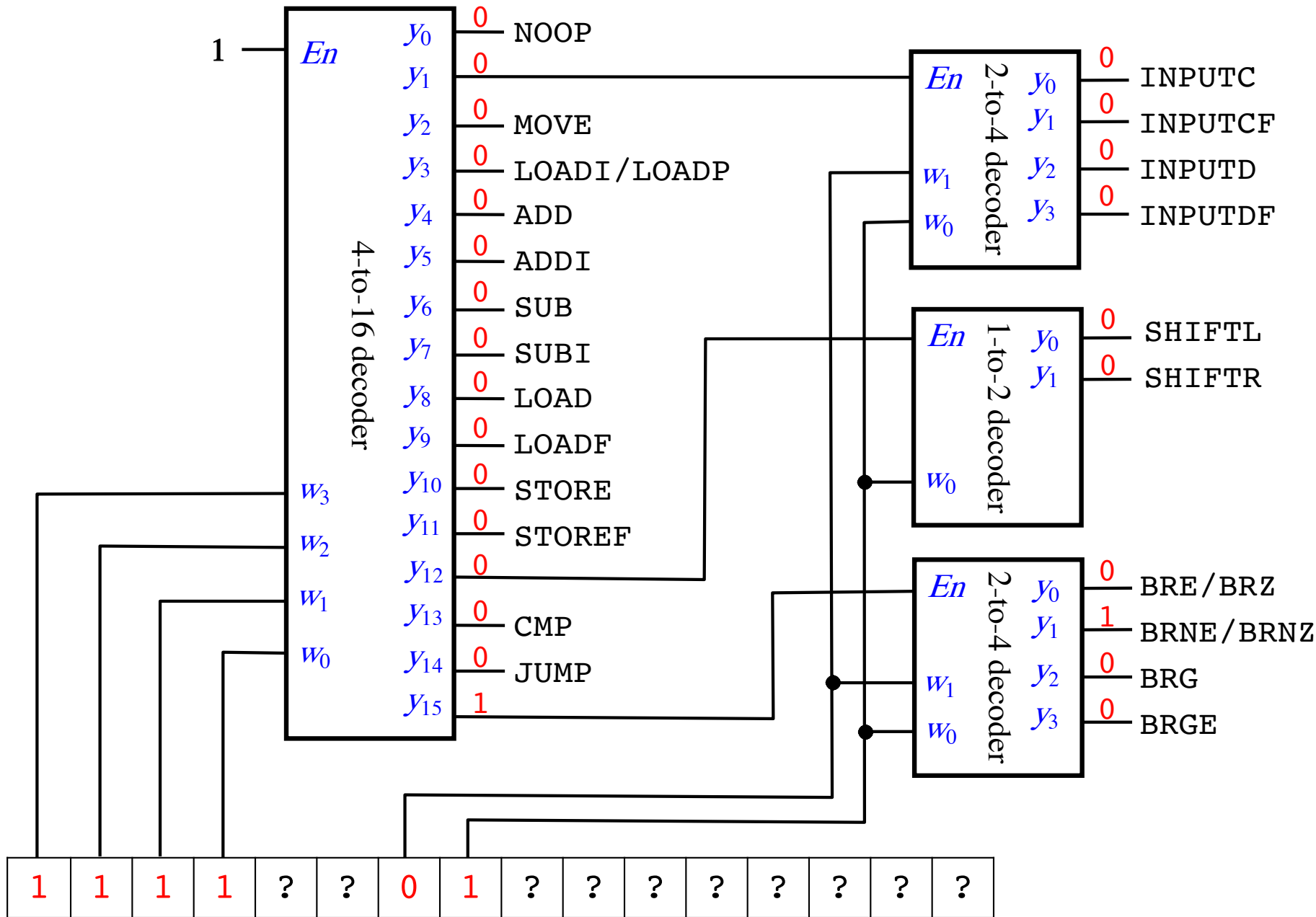
CMP



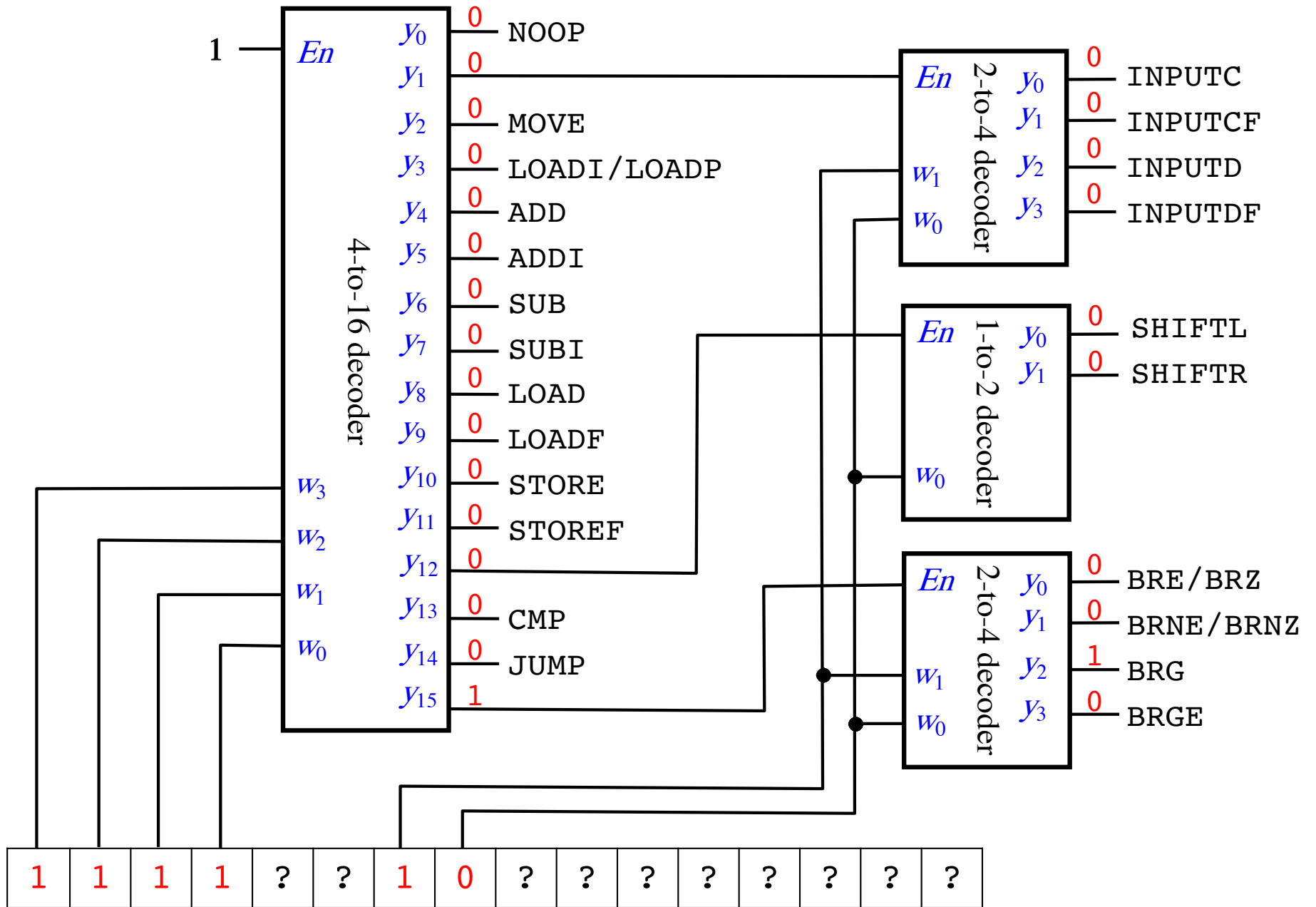
JUMP



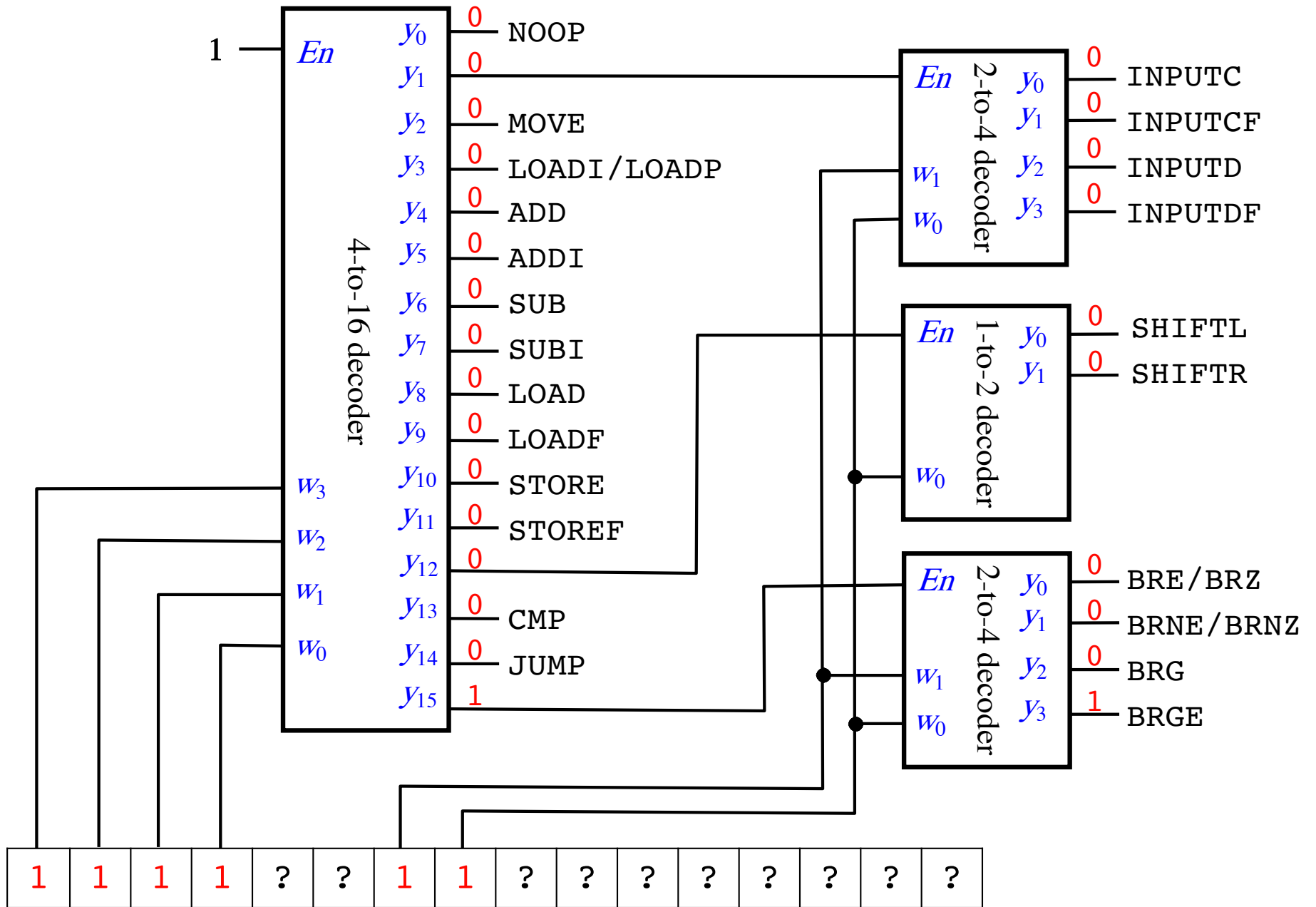
BRE/BRZ



BRNE /BRNZ



BRG



BRGE

The Control Table

	IMEM_WRITE_ENABLE	PROGRAM_COUNTER_MUX	PROGRAM_COUNTER_WRITE_ENABLE	REGISTERS_PORT0_SELECT1	REGISTERS_PORT0_SELECT0	REGISTERS_PORT1_SELECT1	REGISTERS_PORT1_SELECT0	REGISTERS_WRITE_SELECT1	REGISTERS_WRITE_SELECT0	REGISTERS_WRITE_ENABLE	ALU_SOURCE_MUX	ALU_SELECT1	ALU_SELECT0	FLAGS_WRITE_ENABLE	ALU_RESUT_MUX	DMEM_INPUT_MUX	DMEM_WRITE_ENABLE	REG_WRITEBACK_MUX
NOOP			1															
INPUTC	1		1												1			
INPUTCF	1		1	X1	X0						1	1						
INPUTD			1												1	1	1	
INPUTDF			1	X1	X0						1	1				1	1	
MOVE			1	Y1	Y0			X1	X0	1	1	1						
LOADI/LOADP			1					X1	X0	1					1			
ADD			1	X1	X0	Y1	Y0	X1	X0	1		1		1				
ADDI			1	X1	X0			X1	X0	1	1	1		1				
SUB			1	X1	X0	Y1	Y0	X1	X0	1		1	1	1				
SUBI			1	X1	X0			X1	X0	1	1	1	1	1				
LOAD			1					X1	X0	1					1			1
LOADF			1	Y1	Y0			X1	X0	1	1	1						1
STORE			1			X1	X0								1		1	
STOREF			1	Y1	Y0	X1	X0				1	1					1	
SHIFTL			1	X1	X0			X1	X0	1				1				
SHIFTR			1	X1	X0			X1	X0	1			1	1				
CMP			1	X1	X0	Y1	Y0					1	1	1				
JUMP		1	1															
BRE/BRZ		B1	1															
BRNE/BRNZ		B2	1															
BRG		B3	1															
BRGE		B4	1															

Taken from these bits of the instruction

C ₁₅	C ₄	C ₃	C ₂	C ₁	C ₀	C ₉	C ₈	C ₇	C ₆	C ₅	C ₄	C ₃	C ₂	C ₁	C ₀
						Y ₁	Y ₀								

	IMEM_WRITE_ENABLE	PROGRAM_COUNTER_MUX	PROGRAM_COUNTER_WRITE_ENABLE	REGISTERS_PORT0_SELECT1	REGISTERS_PORT0_SELECT0	REGISTERS_PORT1_SELECT1	REGISTERS_PORT1_SELECT0	REGISTERS_WRITE_SELECT1	REGISTERS_WRITE_SELECT0	REGISTERS_WRITE_ENABLE	ALU_SOURCE_MUX	ALU_SELECT1	ALU_SELECT0	FLAGS_WRITE_ENABLE	ALU_RESUT_MUX	DMEM_INPUT_MUX	DMEM_WRITE_ENABLE	REG_WRITEBACK_MUX
NOOP			1															
INPUTC	1		1												1			
INPUTCF	1		1	X1	X0						1	1						
INPUTD			1												1	1	1	
INPUTDF			1	X1	X0						1	1				1	1	
MOVE			1	Y1	Y0			X1	X0	1	1	1						
LOADI/LOADP			1					X1	X0	1					1			
ADD			1	X1	X0	Y1	Y0	X1	X0	1		1		1				
ADDI			1	X1	X0			X1	X0	1	1	1		1				
SUB			1	X1	X0	Y1	Y0	X1	X0	1		1	1	1				
SUBI			1	X1	X0			X1	X0	1	1	1	1	1				
LOAD			1					X1	X0	1					1			1
LOADF			1	Y1	Y0			X1	X0	1	1	1						1
STORE			1			X1	X0								1		1	
STOREF			1	Y1	Y0	X1	X0				1	1					1	
SHIFTL			1	X1	X0			X1	X0	1				1				
SHIFTR			1	X1	X0			X1	X0	1			1	1				
CMP			1	X1	X0	Y1	Y0					1	1	1				
JUMP		1	1															
BRE/BRZ		B1	1															
BRNE/BRNZ		B2	1															
BRG		B3	1															
BRGE		B4	1															

Taken from these bits of the instruction

C ₁₅	C ₄	C ₃	C ₂	C ₁	C ₀	C ₉	C ₈	C ₇	C ₆	C ₅	C ₄	C ₃	C ₂	C ₁	C ₀
						Y ₁	Y ₀								

	IMEM_WRITE_ENABLE	PROGRAM_COUNTER_MUX	PROGRAM_COUNTER_WRITE_ENABLE	REGISTERS_PORT0_SELECT1	REGISTERS_PORT0_SELECT0	REGISTERS_PORT1_SELECT1	REGISTERS_PORT1_SELECT0	REGISTERS_WRITE_SELECT1	REGISTERS_WRITE_SELECT0	REGISTERS_WRITE_ENABLE	ALU_SOURCE_MUX	ALU_SELECT1	ALU_SELECT0	FLAGS_WRITE_ENABLE	ALU_RESUT_MUX	DMEM_INPUT_MUX	DMEM_WRITE_ENABLE	REG_WRITEBACK_MUX
NOOP			1															
INPUTC	1		1												1			
INPUTCF	1		1	X1	X0						1	1						
INPUTD			1												1	1	1	
INPUTDF			1	X1	X0						1	1				1	1	
MOVE			1	Y1	Y0			X1	X0	1	1	1						
LOADI/LOADP			1					X1	X0	1					1			
ADD			1	X1	X0	Y1	Y0	X1	X0	1		1		1				
ADDI			1	X1	X0			X1	X0	1	1	1		1				
SUB			1	X1	X0	Y1	Y0	X1	X0	1		1	1	1				
SUBI			1	X1	X0			X1	X0	1	1	1	1	1				
LOAD			1					X1	X0	1					1			1
LOADF			1	Y1	Y0			X1	X0	1	1	1						1
STORE			1			X1	X0								1		1	
STOREF			1	Y1	Y0	X1	X0				1	1					1	
SHIFTL			1	X1	X0			X1	X0	1				1				
SHIFTR			1	X1	X0			X1	X0	1			1	1				
CMP			1	X1	X0	Y1	Y0					1	1	1				
JUMP		1	1															
BRE/BRZ		B1	1															
BRNE/BRNZ		B2	1															
BRG		B3	1															
BRGE		B4	1															

computed using
the flags register

B1= ZF
 B2= ~ZF
 B3= AND (~ZF, XNOR(NF, OF))
 B4= XNOR(NF, OF)

Zero Flag (ZF)
 Negative Flag (NF)
 Overflow Flag (OF)

Sample Assembly Programs for the i281 CPU

The OPCODEs

(Mapped to Machine Language)

The OPCODEs

NOOP

0	0	0	0	d	d	d	d	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

INPUTC

0	0	0	1	d	d	0	0	C	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

INPUTCF

0	0	0	1	R	X	0	1	C	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

INPUTD

0	0	0	1	d	d	1	0	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

INPUTDF

0	0	0	1	R	X	1	1	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

MOVE

0	0	1	0	R	X	R	Y	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

LOADI/LOADP

0	0	1	1	R	X	d	d	I	M	M	E	D	V	A	L
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

The OPCODEs

ADD

0	1	0	0	R	X	R	Y	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ADDI

0	1	0	1	R	X	d	d	I	M	M	E	D	V	A	L
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

SUB

0	1	1	0	R	X	R	Y	d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

SUBI

0	1	1	1	R	X	d	d	I	M	M	E	D	V	A	L
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

LOAD

1	0	0	0	R	X	d	d	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

LOADF

1	0	0	1	R	X	R	Y	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

STORE

1	0	1	0	R	X	d	d	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

STOREF

1	0	1	1	R	X	R	Y	D	A	D	D	R	E	S	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

The OPCODEs

SHIFTL

1	1	0	0	R	X	d	0	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

SHIFTR

1	1	0	0	R	X	d	1	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

CMP

1	1	0	1	R	X	R	Y	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

JUMP

1	1	1	0	d	d	d	d	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BRE/BRZ

1	1	1	1	d	d	0	0	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BRNE/BRNZ

1	1	1	1	d	d	0	1	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BRG

1	1	1	1	d	d	1	0	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

BRGE

1	1	1	1	d	d	1	1	P	C	O	F	F	S	E	T
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Do Loop

C Version

```
// Add the numbers from 1 to 5 using a do loop.
```

```
int N=5;
```

```
int main()
```

```
{
```

```
    int i, sum;
```

```
    i=0;
```

```
    sum=0;
```

```
    do
```

```
    {
```

```
        i++;
```

```
        sum+=i;
```

```
    }while( i < N );
```

```
}
```

Assembly Version

```
; Add the numbers from 1 to 5 using a do loop.
```

```
.data
```

```
N      BYTE      5
```

```
sum    BYTE      ?
```

```
.code
```

```
    LOADI A, 0      ; i = 0
```

```
    LOADI B, 0      ; sum=0
```

```
    LOAD D, [N]     ; register D = N
```

```
Do:  ADDI A, 1      ; i++
```

```
    ADD B, A        ; sum+=i
```

```
    CMP D, A        ; N > i ? (register ordering is swapped)
```

```
    BRG Do          ; if true, jump to Do
```

```
End:  STORE [sum], B ; store sum to memory
```

```
; Register allocation:
```

```
; A: i (the variable i is optimized to register A)
```

```
; B: sum
```

```
; C: <not used>
```

```
; D: N
```

Machine Code Version

Data Memory:

00000101

00000000

Code Memory:

0011000000000000

0011010000000000

1000110000000000

0101000000000001

0100010000000000

1101110000000000

1111001011111100

1010010000000001

Assembly v.s. Machine Code

.data

N BYTE 5

sum BYTE ?

.code

LOADI A, 0

LOADI B, 0

LOAD D, [N]

Do: ADDI A, 1

ADD B, A

CMP D, A

BRG Do

End: STORE [sum], B

Data Memory:

00000101

00000000

Code Memory:

0011000000000000

0011010000000000

1000110000000000

0101000000000001

0100010000000000

1101110000000000

1111001011111100

1010010000000001

Assembly v.s. Machine Code

.data

N BYTE 5

sum BYTE ?

.code

LOADI A, 0

LOADI B, 0

LOAD D, [N]

Do: ADDI A, 1

ADD B, A

CMP D, A

BRG Do

End: STORE [sum], B

Data Memory:

00000101

00000000

Code Memory:

00110000_00000000

00110100_00000000

10001100_00000000

01010000_00000001

01000100_00000000

11011100_00000000

11110010_11111100

10100100_00000001

Assembly v.s. Machine Code

.data

N BYTE 5

sum BYTE ?

.code

LOADI A, 0

LOADI B, 0

LOAD D, [N]

Do: ADDI A, 1

ADD B, A

CMP D, A

BRG Do

End: STORE [sum], B

Data Memory:

00000101

00000000

Code Memory:

0011_00_00_00000000

0011_01_00_00000000

1000_11_00_00000000

0101_00_00_00000001

0100_01_00_00000000

1101_11_00_00000000

1111_00_10_11111100

1010_01_00_00000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**
sum **BYTE** **?**

Data Memory:

00000101
00000000

.code

LOADI **A, 0**
 LOADI **B, 0**
 LOAD **D, [N]**
Do: **ADDI** **A, 1**
 ADD **B, A**
 CMP **D, A**
 BRG **Do**
End: **STORE** **[sum], B**

Code Memory:

0011_00_00_00000000
0011_01_00_00000000
1000_11_00_00000000
0101_00_00_00000001
0100_01_00_00000000
1101_11_00_00000000
1111_00_10_11111100
1010_01_00_00000001

Assembly v.s. Machine Code

.data

N BYTE 5

sum BYTE ?

.code

LOADI A, 0

LOADI B, 0

LOAD D, [N]

Do: ADDI A, 1

ADD B, A

CMP D, A

BRG Do

End: STORE [sum], B

Data Memory:

00000101

00000000

Code Memory:

0011_00_00_00000000

0011_01_00_00000000

1000_11_00_00000000

0101_00_00_00000001

0100_01_00_00000000

1101_11_00_00000000

1111_00_10_11111100

1010_01_00_00000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**

sum **BYTE** **?**

.code

LOADI **A, 0**

LOADI **B, 0**

LOAD **D, [N]**

Do: **ADDI** **A, 1**

ADD **B, A**

CMP **D, A**

BRG **Do**

End: **STORE** **[sum], B**

Data Memory:

00000101

00000000

Code Memory:

0011_00_00_00000000

0011_01_00_00000000

1000_11_00_00000000

0101_00_00_00000001

0100_01_00_00000000

1101_11_00_00000000

1111_00_10_11111100

1010_01_00_00000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**

sum **BYTE** **?**

.code

LOADI **A**, **0**

LOADI **B**, **0**

LOAD **D**, [**N**]

Do: **ADDI** **A**, **1**

ADD **B**, **A**

CMP **D**, **A**

BRG **Do**

End: **STORE** [**sum**], **B**

Data Memory:

00000101

00000000

Code Memory:

0011_00_00_00000000

0011_01_00_00000000

1000_11_00_00000000

0101_00_00_00000001

0100_01_00_00000000

1101_11_00_00000000

1111_00_10_11111100

1010_01_00_00000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**
sum **BYTE** **?**

Data Memory:

00000101
00000000

.code

LOADI **A**, **0**
LOADI **B**, **0**
LOAD **D**, [**N**]
Do: **ADDI** **A**, **1**
 ADD **B**, **A**
 CMP **D**, **A**
 BRG **Do**
End: **STORE** [**sum**], **B**

Code Memory:

0011_00_00_00000000
0011_01_00_00000000
1000_11_00_00000000
0101_00_00_00000001
0100_01_00_00000000
1101_11_00_00000000
1111_00_10_11111100
1010_01_00_00000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**
sum **BYTE** **?**

Data Memory:

00000101
00000000

.code

LOADI **A**, **0**
LOADI **B**, **0**
LOAD **D**, [**N**]
Do: **ADDI** **A**, **1**
 ADD **B**, **A**
 CMP **D**, **A**
 BRG **Do**
End: **STORE** [**sum**], **B**

Code Memory:

0011_00_00_00000000
0011_01_00_00000000
1000_11_00_00000000
0101_00_00_00000001
0100_01_00_00000000
1101_11_00_00000000
1111_00_10_11111100
1010_01_00_00000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**
sum **BYTE** **?**

Data Memory:

00000101
00000000

.code

LOADI **A**, **0**
LOADI **B**, **0**
LOAD **D**, [**N**]
Do: **ADDI** **A**, **1**
 ADD **B**, **A**
 CMP **D**, **A**
 BRG **Do**
End: **STORE** [**sum**], **B**

Code Memory:

0011_00_dd_00000000
0011_01_dd_00000000
1000_11_dd_00000000
0101_00_dd_00000001
0100_01_00_dddddddd
1101_11_00_dddddddd
1111_dd_10_11111100
1010_01_dd_00000001

Assembly v.s. Machine Code

.data

N **BYTE** **5**
sum **BYTE** **?**

Data Memory:

00000101
00000000

.code

LOADI **A, 0**
 LOADI **B, 0**
 LOAD **D, [N]**
Do: **ADDI** **A, 1**
 ADD **B, A**
 CMP **D, A**
 BRG **Do**
End: **STORE** **[sum], B**

Code Memory:

0011_00_00_00000000
0011_01_00_00000000
1000_11_00_00000000
0101_00_00_00000001
0100_01_00_00000000
1101_11_00_00000000
1111_00_10_11111100
1010_01_00_00000001

Bubble Sort

C Version

```
int array[] = {7, 3, 2, 1, 6, 4, 5, 8};
int last = 7; // last valid index in the array
int temp;
int i, j;

int main()
{
    for (i = 0; i < last; i++)
        for (j = 0; j < last-i; j++)
            if (array[j] > array[j+1]){
                temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
            }

    //for(i = 0; i < N; i++){
    //    printf("%d, ", array[i]);
    //}
}
```

C Version

```
int array[] = {7, 3, 2, 1, 6, 4, 5, 8};
int last = 7; // last valid index in the array
int temp;
int i, j;

int main()
{
    for (i = 0; i < last; i++)
        for (j = 0; j < last-i; j++)
            if (array[j] > array[j+1]){
                temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
            }

    //for(i = 0; i < N; i++){
    //    printf("%d, ", array[i]);
    //}
}
```

Assembly Version

```
.data
array  BYTE 7, 3, 2, 1, 6, 4, 5, 8
last   BYTE 7
temp   BYTE ?

.code

        LOADI  A, 0                ; i = 0;
Outer:  LOAD   D, [last]           ; Load last into D
        LOADI  B, 0                ; j = 0;
        CMP   A, D                 ; i < last
        BRGE  End                 ; If i >= last break out of the outer loop
Inner:  LOAD   D, [last]           ; Re-Load last into D (this register is shared)
        SUB   D, A                 ; D = D - A (i.e., D = last - i)
        CMP   B, D                 ; j < last - i
        BRGE  Iinc                ; If j >= last-i branch to Iinc
If:     LOADF  C, [array+B]        ; C = array[j]
        LOADF  D, [array+B+1]      ; D = array[j+1] (compiler adds 1 to addr. of array)
        CMP   D, C                 ; if array[j+1] < array[j] (switched direction)
        BRGE  Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI  B, 1                 ; j++
        JUMP  Inner
Iinc:   ADDI  A, 1                 ; i++
        JUMP  Outer
End:    NOOP                       ; Do nothing

; Register allocation:
; A: i
; B: j
; C: array[j]
; D: last, array[j+1]

; Notes: i and j are optimized away. They exist only in registers, not in the main memory.
```

Assembly Version

```
.data
array  BYTE 7, 3, 2, 1, 6, 4, 5, 8
last   BYTE 7
temp   BYTE ?

.code

        LOADI  A, 0                ; i = 0;
Outer:  LOAD   D, [last]           ; Load last into D
        LOADI  B, 0                ; j = 0;
        CMP    A, D                ; i < last
        BRGE   End                ; If i >= last break out of the outer loop
Inner:  LOAD   D, [last]           ; Re-Load last into D (this register is shared)
        SUB    D, A                ; D = D - A (i.e., D = last - i)
        CMP    B, D                ; j < last - i
        BRGE   Iinc               ; If j >= last-i branch to Iinc
If:     LOADF  C, [array+B]        ; C = array[j]
        LOADF  D, [array+B+1]      ; D = array[j+1] (compiler adds 1 to addr. of array)
        CMP    D, C                ; if array[j+1] < array[j] (switched direction)
        BRGE   Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1                ; j++
        JUMP   Inner
Iinc:   ADDI   A, 1                ; i++
        JUMP   Outer
End:    NOOP                       ; Do nothing

; Register allocation:
; A: i
; B: j
; C: array[j]
; D: last, array[j+1]

; Notes: i and j are optimized away. They exist only in registers, not in the main memory.
```


Machine Code Version

```
.data
array  BYTE 7, 3, 2, 1, 6, 4, 5, 8
last   BYTE 7
temp   BYTE ?
```

```
.code

        LOADI  A, 0
Outer:  LOAD   D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE  End
Inner:  LOAD   D, [last]
        SUB    D, A
        CMP    B, D
        BRGE  Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE  Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP  Inner
Iinc:   ADDI   A, 1
        JUMP  Outer
End:    NOOP
```


Machine Code Version

```
.data
array  BYTE 7, 3, 2, 1, 6, 4, 5, 8
last   BYTE 7
temp   BYTE ?
```

```
.code
```

```
Outer:  LOADI  A, 0
        LOAD  D, [last]
        LOADI B, 0
        CMP   A, D
        BRGE  End
Inner:  LOAD  D, [last]
        SUB   D, A
        CMP   B, D
        BRGE  Iinc
If:     LOADF C, [array+B]
        LOADF D, [array+B+1]
        CMP   D, C
        BRGE  Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI  B, 1
        JUMP  Inner
Iinc:   ADDI  A, 1
        JUMP  Outer
End:    NOOP
```

Data Memory:

```
00000111
00000011
00000010
00000001
00000110
00000100
00000101
00001000
00000111
00000000
```

Machine Code Version

```
.data
array  BYTE 7, 3, 2, 1, 6, 4, 5, 8
last   BYTE 7
temp   BYTE ?

.code

        LOADI  A, 0
Outer:  LOAD    D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE  End
Inner:  LOAD    D, [last]
        SUB    D, A
        CMP    B, D
        BRGE  Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE  Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP  Inner
Iinc:   ADDI   A, 1
        JUMP  Outer
End:    NOOP
```

Data Memory:

```
00000111 //array[0]
00000011 //array[1]
00000010 //array[2]
00000001 //array[3]
00000110 //array[4]
00000100 //array[5]
00000101 //array[6]
00001000 //array[7]
00000111 //last
00000000 //temp
```

Machine Code Version

		Address	Data Memory:
.data			
array	BYTE 7, 3, 2, 1, 6, 4, 5, 8	0000	00000111 //array[0]
last	BYTE 7	0001	00000011 //array[1]
temp	BYTE ?	0010	00000010 //array[2]
		0011	00000001 //array[3]
.code		0100	00000110 //array[4]
Outer:	LOADI A, 0	0101	00000100 //array[5]
	LOAD D, [last]	0110	00000101 //array[6]
	LOADI B, 0	0111	00001000 //array[7]
	CMP A, D	1000	00000111 //last
	BRGE End	1001	00000000 //temp
Inner:	LOAD D, [last]		
	SUB D, A		
	CMP B, D		
	BRGE Iinc		
If:	LOADF C, [array+B]		
	LOADF D, [array+B+1]		
	CMP D, C		
	BRGE Jinc		
Swap:	STOREF [array+B], D		
	STOREF [array+B+1], C		
Jinc:	ADDI B, 1		
	JUMP Inner		
Iinc:	ADDI A, 1		
	JUMP Outer		
End:	NOOP		

Machine Code Version

		Address	Data Memory:
.data			
array	BYTE 7, 3, 2, 1, 6, 4, 5, 8	0000	00000111 //array[0]
last	BYTE 7	0001	00000011 //array[1]
temp	BYTE ?	0010	00000010 //array[2]
		0011	00000001 //array[3]
.code		0100	00000110 //array[4]
Outer:	LOADI A, 0	0101	00000100 //array[5]
	LOAD D, [last]	0110	00000101 //array[6]
	LOADI B, 0	0111	00001000 //array[7]
	CMP A, D	1000	00000111 //last
	BRGE End	1001	00000000 //temp
Inner:	LOAD D, [last]	1010	00000000
	SUB D, A	1011	00000000
	CMP B, D	1100	00000000
	BRGE Iinc	1101	00000000
If:	LOADF C, [array+B]	1110	00000000
	LOADF D, [array+B+1]	1111	00000000
	CMP D, C		
	BRGE Jinc		
Swap:	STOREF [array+B], D		
	STOREF [array+B+1], C		
Jinc:	ADDI B, 1		
	JUMP Inner		
Iinc:	ADDI A, 1		
	JUMP Outer		
End:	NOOP		

Machine Code Version

```
.data
array  BYTE 7, 3, 2, 1, 6, 4, 5, 8
last   BYTE 7
temp   BYTE ?
```

```
.code

        LOADI  A, 0
Outer:  LOAD   D, [last]
        LOADI  B, 0
        CMP    A, D
        BRGE  End
Inner:  LOAD   D, [last]
        SUB    D, A
        CMP    B, D
        BRGE  Iinc
If:     LOADF  C, [array+B]
        LOADF  D, [array+B+1]
        CMP    D, C
        BRGE  Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1
        JUMP  Inner
Iinc:   ADDI   A, 1
        JUMP  Outer
End:    NOOP
```

Machine Code Version

```
.data
array  BYTE 7, 3, 2, 1, 6, 4, 5, 8
last   BYTE 7
temp   BYTE ?
```

```
.code

Outer:  LOADI  A, 0
        LOAD  D, [last]
        LOADI B, 0
        CMP   A, D
        BRGE  End
Inner:  LOAD  D, [last]
        SUB   D, A
        CMP   B, D
        BRGE  Iinc
If:     LOADF C, [array+B]
        LOADF D, [array+B+1]
        CMP   D, C
        BRGE  Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI  B, 1
        JUMP  Inner
Iinc:   ADDI  A, 1
        JUMP  Outer
End:    NOOP
```

Code Memory:

```
0011000000000000
1000110000001000
0011010000000000
1101001100000000
1111001100001110
1000110000001000
0110110000000000
1101011100000000
1111001100001000
1001100100000000
1001110100000001
1101111000000000
1111001100000010
1011110100000000
1011100100000001
0101010000000001
1110000011110100
0101000000000001
1110000011101110
0000000000000000
```

Machine Code Version

```
.data
array  BYTE 7, 3, 2, 1, 6, 4, 5, 8
last   BYTE 7
temp   BYTE ?
```

```
.code
                                Address  Code Memory:
Outer:  LOADI  A, 0                100000  0011000000000000
        LOAD   D, [last]           100001  1000110000001000
        LOADI  B, 0                100010  0011010000000000
        CMP    A, D                100011  1101001100000000
        BRGE   End                 100100  1111001100001110
Inner:  LOAD   D, [last]           100101  1000110000001000
        SUB    D, A                100110  0110110000000000
        CMP    B, D                100111  1101011100000000
        BRGE   Iinc               101000  1111001100001000
If:     LOADF  C, [array+B]         101001  1001100100000000
        LOADF  D, [array+B+1]     101010  1001110100000001
        CMP    D, C                101011  1101111000000000
        BRGE   Jinc               101100  1111001100000010
Swap:   STOREF [array+B], D        101101  1011110100000000
        STOREF [array+B+1], C    101110  1011100100000001
Jinc:   ADDI   B, 1                101111  0101010000000001
        JUMP   Inner              110000  1110000011110100
Iinc:   ADDI   A, 1                110001  0101000000000001
        JUMP   Outer              110010  1110000011101110
End:    NOOP                       110011  0000000000000000
```

Machine Code Version

```
.data
array  BYTE 7, 3, 2, 1, 6, 4, 5, 8
last   BYTE 7
temp   BYTE ?
```

		Address	Code Memory:
	LOADI A, 0	100000	0011000000000000
Outer:	LOAD D, [last]	100001	1000110000001000
	LOADI B, 0	100010	0011010000000000
	CMP A, D	100011	1101001100000000
	BRGE End	100100	1111001100001110
Inner:	LOAD D, [last]	100101	1000110000001000
	SUB D, A	100110	0110110000000000
	CMP B, D	100111	1101011100000000
	BRGE Iinc	101000	1111001100001000
If:	LOADF C, [array+B]	101001	1001100100000000
	LOADF D, [array+B+1]	101010	1001110100000001
	CMP D, C	101011	1101111000000000
	BRGE Jinc	101100	1111001100000010
Swap:	STOREF [array+B], D	101101	1011110100000000
	STOREF [array+B+1], C	101110	1011100100000001
Jinc:	ADDI B, 1	101111	0101010000000001
	JUMP Inner	110000	1110000011110100
Iinc:	ADDI A, 1	110001	0101000000000001
	JUMP Outer	110010	1110000011101110
End:	NOOP	110011	0000000000000000
		110100	0000000000000000
	
		111110	0000000000000000
		111111	0000000000000000

Assembly v.s. Machine Code

		Code Memory:
.code		0011000000000000
	LOADI A, 0	
Outer:	LOAD D, [last]	1000110000001000
	LOADI B, 0	0011010000000000
	CMP A, D	1101001100000000
	BRGE End	1111001100001110
Inner:	LOAD D, [last]	1000110000001000
	SUB D, A	0110110000000000
	CMP B, D	1101011100000000
	BRGE Iinc	1111001100001000
If:	LOADF C, [array+B]	1001100100000000
	LOADF D, [array+B+1]	1001110100000001
	CMP D, C	1101111000000000
	BRGE Jinc	1111001100000010
Swap:	STOREF [array+B], D	1011110100000000
	STOREF [array+B+1], C	1011100100000001
Jinc:	ADDI B, 1	0101010000000001
	JUMP Inner	1110000011110100
Iinc:	ADDI A, 1	0101000000000001
	JUMP Outer	1110000011101110
End:	NOOP	0000000000000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	00110000_00000000
Outer:	LOAD D, [last]	10001100_00001000
	LOADI B, 0	00110100_00000000
	CMP A, D	11010011_00000000
	BRGE End	11110011_00001110
Inner:	LOAD D, [last]	10001100_00001000
	SUB D, A	01101100_00000000
	CMP B, D	11010111_00000000
	BRGE Iinc	11110011_00001000
If:	LOADF C, [array+B]	10011001_00000000
	LOADF D, [array+B+1]	10011101_00000001
	CMP D, C	11011110_00000000
	BRGE Jinc	11110011_00000010
Swap:	STOREF [array+B], D	10111101_00000000
	STOREF [array+B+1], C	10111001_00000001
Jinc:	ADDI B, 1	01010100_00000001
	JUMP Inner	11100000_11110100
Iinc:	ADDI A, 1	01010000_00000001
	JUMP Outer	11100000_11101110
End:	NOOP	00000000_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		0011_00_00_00000000
	LOADI A, 0	
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_dd_00000000
Outer:	LOAD D, [last]	1000_11_dd_00001000
	LOADI B, 0	0011_01_dd_00000000
	CMP A, D	1101_00_11_dddddddd
	BRGE End	1111_dd_11_00001110
Inner:	LOAD D, [last]	1000_11_dd_00001000
	SUB D, A	0110_11_00_dddddddd
	CMP B, D	1101_01_11_dddddddd
	BRGE Iinc	1111_dd_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_dddddddd
	BRGE Jinc	1111_dd_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_dd_00000001
	JUMP Inner	1110_dd_dd_11110100
Iinc:	ADDI A, 1	0101_00_dd_00000001
	JUMP Outer	1110_dd_dd_11101110
End:	NOOP	0000_dd_dd_dddddddd

Assembly v.s. Machine Code

		Code Memory:
.code		
	LOADI A, 0	0011_00_00_00000000
Outer:	LOAD D, [last]	1000_11_00_00001000
	LOADI B, 0	0011_01_00_00000000
	CMP A, D	1101_00_11_00000000
	BRGE End	1111_00_11_00001110
Inner:	LOAD D, [last]	1000_11_00_00001000
	SUB D, A	0110_11_00_00000000
	CMP B, D	1101_01_11_00000000
	BRGE Iinc	1111_00_11_00001000
If:	LOADF C, [array+B]	1001_10_01_00000000
	LOADF D, [array+B+1]	1001_11_01_00000001
	CMP D, C	1101_11_10_00000000
	BRGE Jinc	1111_00_11_00000010
Swap:	STOREF [array+B], D	1011_11_01_00000000
	STOREF [array+B+1], C	1011_10_01_00000001
Jinc:	ADDI B, 1	0101_01_00_00000001
	JUMP Inner	1110_00_00_11110100
Iinc:	ADDI A, 1	0101_00_00_00000001
	JUMP Outer	1110_00_00_11101110
End:	NOOP	0000_00_00_00000000

Questions?

THE END