

Suggested exercises: Complete before the final exam

- Convert the following floating point numbers to their decimal equivalents (IEEE 32 bit format):

```

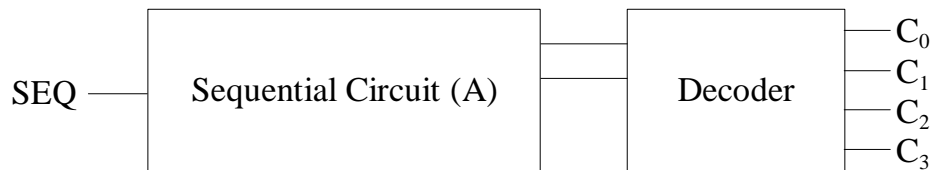
0 1 0 1 1 0 0 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 1 1 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    
```

- Add the two numbers given in 1 and express in the IEEE format.
- Multiply the two numbers given in 1 and express in the IEEE format.
- Divide the two numbers given in 1 and express in the same format.
- Convert 3412.341 to IEEE 32 bit format.
- Convert -3412.341 to IEEE 64 bit format.
- Create an RTL program (using the syntax of the manual in the lecture supplement) to manipulate a 24-bit register as a stack of 3 8-bit values. From INBUS, "push" an 8-bit value onto the stack. "pop" the stack onto OUTBUS (removing the stacktop). Note "push" will require an 8-bit shift in one direction and "pop" an 8-bit shift in the other. Utilize a 6-bit status register as follows:
 - Bits 0 and 1 are user control bits and the remaining bits are for internal control of the stack
 - User bit 0 flags whether the operation to perform is "push" or "pop"
 - User bit 1 flags whether or not the user is maintaining a stack extension beyond the 3 bytes in the register.
 - Bits 2 and 3 monitor register stack size (0, 1, 2, or 3 bytes)
 - On overflow (a "push" with register stack size of 3 bytes), if user bit 1 is "off", don't "push" the new data; if user bit 1 is "on", move the bottom of the stack onto OUTBUS before "push" of the new data
 - On "pop", if the stack is full and bit 1 is "on", move INBUS onto the bottom of the stack after "pop" of the stack.

You may assume in your RTL code that the registers are already loaded (so you don't have to weave in any I/O).

- Consider the following control sequence and associated circuit:

- $A \leftarrow B, B \leftarrow C, C \leftarrow A$
 $\rightarrow 2 \cdot \text{SEQ} + 3 \cdot \overline{\text{SEQ}}$
- $B \leftarrow A + C$
 $\rightarrow 2 \cdot \text{SEQ} + 0 \cdot \overline{\text{SEQ}}$
- $A \leftarrow 0$
 $\rightarrow 0 \cdot \text{SEQ} + 3 \cdot \overline{\text{SEQ}}$
- $A \leftarrow 1, B \leftarrow 0$
 $\rightarrow 0 \cdot \text{SEQ} + 1 \cdot \overline{\text{SEQ}}$

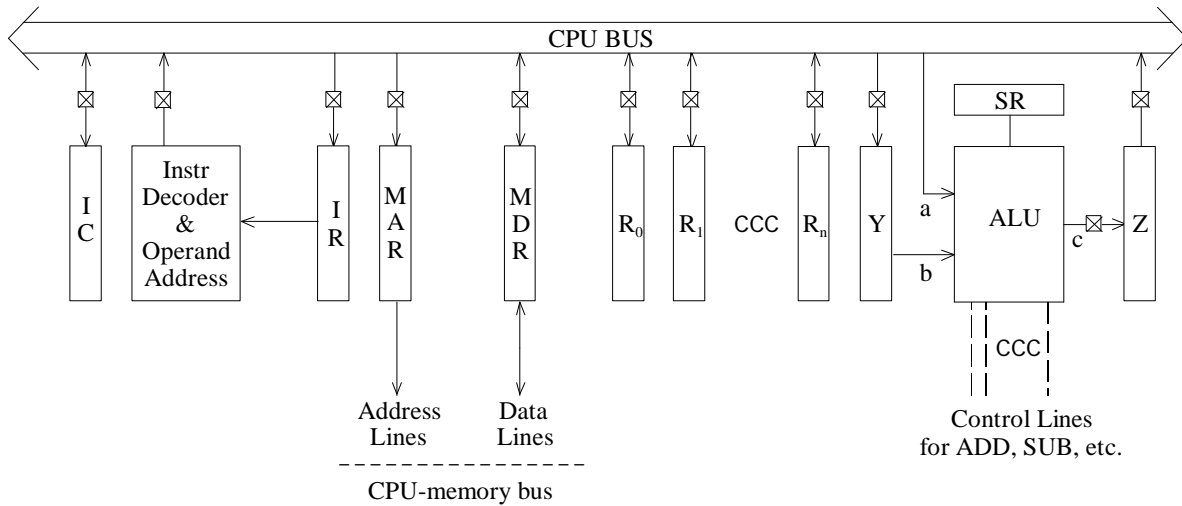


Construct the sequential circuit (A) to process the control sequence.

Note that the control sequence description tells you how to determine the next state (C0, C1, C2, or C3) given the current state and the input sequencing signal SEQ.

9. Recall the following CPU architecture.

Single Bus CPU Organization Example



Construct a microcode sequence for

BGT* A

to load the IC indirect from the address stored at the memory location given by A (i.e., the address of the item to load is not given by A directly, but is given by the value stored at the memory location A references) if the most recent "compare" set the status register (SR) for GT.

10. Referencing the same architecture as in problem 9, devise a microcode sequence for executing each of:

LDXR₁ A

and

MOVE01

Instruction specifications:

For LDXR₁

The address of the item to load into R₁ is obtained from the memory location given by A modified by R₀ as an index (i.e., you must add R₀ to the address to get the address of the item to load).

For MOVE01

The action is simply to copy the contents of the memory location addressed by register R₀ to the memory location addressed by register R₁.

7. Create an RTL program (using the syntax of the manual in the lecture supplement you acquired from the bookstore) to manipulate a 24-bit register as a stack of 3 8-bit values. From INBUS, "push" an 8-bit value onto the stack. "pop" the stack onto OUTBUS (removing the stacktop). Note "push" will require an 8-bit shift in one direction and "pop" an 8-bit shift in the other. Utilize a 6-bit status register as follows:

Bits 0 and 1 are user control bits and the remaining bits are for internal control of the stack

User bit 0 flags whether the operation to perform is "push" or "pop"

User bit 1 flags whether or not the user is maintaining a stack extension beyond the 3 bytes in the register.

Bits 2 and 3 monitor register stack size (0, 1, 2, or 3 bytes)

On overflow (a "push" with register stack size of 3 bytes), if user bit 1 is "off", don't "push" the new data; if user bit 1 is "on", move the bottom of the stack onto OUTBUS before "push" of the new data

On "pop", if the stack is full and bit 1 is "on", move INBUS onto the bottom of the stack after "pop" of the stack.

You may assume in your RTL code that the registers are already loaded (so you don't have to weave in any I/O).

```
[0] RtlSTACK
[1] DEFREG:
[2] STACK(24)
[3] STATUS(6)
[4] DEFBUS:
[5] BEGIN:
** STATUS[0] = 1 for POP, 0 for PUSH
[6] (STATUS[0]) MERGEAT POP
** Stack is full if STATUS[2 3] = 1 1
[7] (STATUS[1 2 3] LEQ 0 1 1) MERGEAT QUIT
** STATUS[1] = 1 for user stack extension, 0 for none
[8] (STATUS[1 2 3] LEQ 1 1 1) OUTBUS SETBUS STACK[16 TO 23]
** The actual PUSH is done by lines 9, 10
[9] STACK SETREG 8 RLSHIFT STACK
[10] STACK[0 TO 7] SETREG 8 INBUS 'Enter 8 bit value to PUSH'
[11] (STATUS[2 3] NLEQ 1 1) STATUS[2 3] SETREG 0 1 ADD STATUS[2 3]
[12] MERGEAT QUIT
** If stack is not empty, the POP to OUTBUS is done by lines 14, 15
[13] POP:(STATUS[2 3] LEQ 0 0) MERGEAT QUIT
[14] OUTBUS SETBUS STACK[0 TO 7]
[15] STACK SETREG 8 LLSHIFT STACK
[16] (STATUS[1 2 3] LEQ 1 1 1) STACK[16 TO 23] SETREG
      8 INBUS '8 from usr stack'
[17] (STATUS[1 2 3] NLEQ 1 1 1) STATUS[2 3] SETREG
      STATUS[2 3] SUB 0 1
[18] QUIT:
[19] END:
```

8. Consider the following control sequence and associated circuit:

0. $A \leftarrow B, B \leftarrow C, C \leftarrow A$

$\rightarrow 2 \bullet \text{SEQ} + 3 \bullet \overline{\text{SEQ}}$

1. $B \leftarrow A + C$

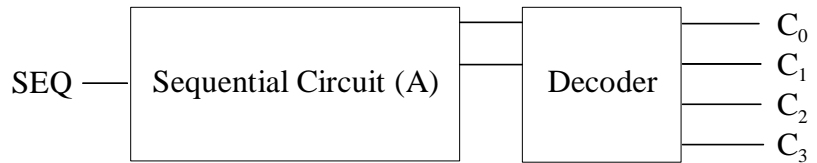
$\rightarrow 2 \bullet \text{SEQ} + 0 \bullet \overline{\text{SEQ}}$

2. $A \leftarrow 0$

$\rightarrow 0 \bullet \text{SEQ} + 3 \bullet \overline{\text{SEQ}}$

3. $A \leftarrow 1, B \leftarrow 0$

$\rightarrow 0 \bullet \text{SEQ} + 1 \bullet \overline{\text{SEQ}}$



Construct the sequential circuit (A) to process the control sequence.

Note that the control sequence description tells you how to determine the next state ($C_0, C_1, C_2,$ or C_3) given the current state and the input sequencing signal SEQ .

SEQ	Q_1	Q_0	$Q_1^n = D_1$	$Q_0^n = D_0$
0	0	0	1	1
0	0	1	0	0
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	0	0
1	1	1	0	0

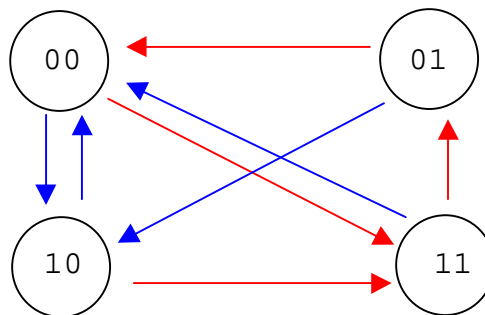
$$D_1 = \overline{\text{SEQ}} \cdot \overline{Q_0} + \text{SEQ} \cdot \overline{Q_1}$$

$$D_0 = \overline{\text{SEQ}} \cdot \overline{Q_0} + \overline{\text{SEQ}} \cdot Q_1$$

<State diagram exhibiting control flow among states 0, 1, 2, 3>

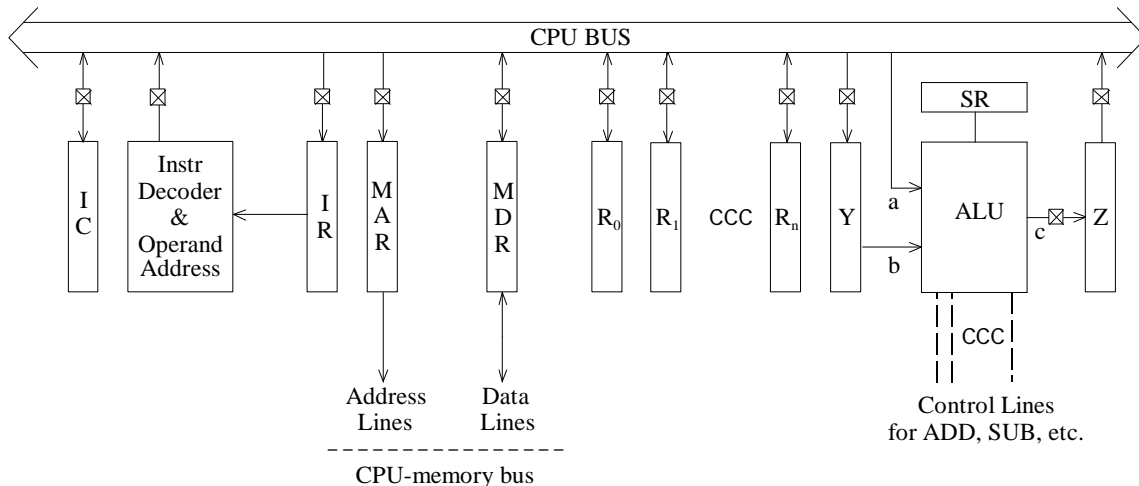
<K-map analysis to obtain equations for D_0 and D_1 >

0 = red 1 = blue



9. Recall the following CPU architecture.

Single Bus CPU Organization Example



Construct a microcode sequence for

BGT* A

to load the IC indirect from the address stored at the memory location given by A (i.e., the address of the item to load is not given by A directly, but is given by the value stored at the memory location A references) if the most recent "compare" set the status register (SR) for GT.

<standard instruction fetch here>

$Addr_{out}, MAR_{in}, Read, WaitM$

[go to memory to get the branch address]

MDR_{out}, Y_{in}

[branch address in Y for case SR is 1 1]

IC_{out}, GT, Z_{in}

[if SR is 1 1, GT picks Y, otherwise IC from bus]

Z_{out}, IC_{in}

[send result of GT back to IC]

10. Referencing the same architecture as in problem 9, devise a microcode sequence for executing

each of: $LDXR_1 A$

and $MOVE01$

Instruction specifications:

For $LDXR_1$: The address of the item to load into R_1 is obtained from the memory location given by A modified by R_0 as an index (i.e., you must add R_0 to the address to get the address of the item to load).

For $MOVE01$: The action is simply to copy the contents of the memory location addressed by register R_0 to the memory location addressed by register R_1 .

$LDXR1 <addr>$: *<standard instruction fetch here>*

$Addr_{out}, Y_{in}$

[move the address part of the instruction (A) to Y]

$R0_{out}, Add, Z_{in}$

[add on the value of the index register R0]

$Z_{out}, MAR_{in}, Read, WaitM$

[read from memory using the modified address]

$MDR_{out}, R1_{in}$

[store the retrieved value in R1]

$MOVE01$: *<standard instruction fetch here>*

$R0_{out}, MAR_{in}, Read, WaitM$

[read from memory using the address in R0]

$R1_{out}, MAR_{in}, Write, WaitM$

[once data is in MDR, write to memory using the address in R1]