EECE 2322: Fundamentals of Digital Design and Computer Organization

Lecture 11_2: Midterm Review and FSM

Xiaolin Xu
Department of ECE
Northeastern University

Midterm2 Review

- Topics:
  - MIPS instruction basics
    - E.g., what does a line of assembly code do, and its results?

From High-level Code to Assembly Language

C Code

```
a = b + c;
f= (g + h) - (i + j)
```

MIPS assembly code

```
# assuming that a-to-c are stored in $s0-
   to-$s2, f-to—j are stored in $s3-to-$s7


# $s0 = a, $s1 = b, $s2 = c, $s3 = f, $s4 =
   g, $s5 = h # $s6 = i, $s7 = j
```

How to convert the C code to assembly code?

```
_____________________ #a=b-c
_____________________#$t0=g+h
_____________________#$t1=i+j
_____________________#f=(g+h)-(i+j)
```

From High-level Code to Assembly Language

C Code

```
a = b + c;
f= (g + h) - (i + j)
```

MIPS assembly code

```
# assuming that a-to-c are stored in $s0-
   to-$s2, f-to—j are stored in $s3-to-$s7

# $s0 = a, $s1 = b, $s2 = c, $s3 = f, $s4 =
   g, $s5 = h # $s6 = i, $s7 = j
```

How to convert the C code to assembly code?

```
sub$s0,$s1,$s2 #a=b-c                    _____________________ #a=b-c
add$t0,$s4,$s5 #$t0=g+h                  _____________________#$t0=g+h
add$t1,$s6,$s7 #$t1=i+j                  _____________________#$t1=i+j
sub$s3,$t0,$t1 #f=(g+h)-(i+j)            _____________________#f=(g+h)-(i+j)
```

Conditional Branching (beq)

 # MIPS assembly **Results per line?**

```
 addi $s0, $0, 4
 addi $s1, $0, 1
 sll  $s1, $s1, 2
 beq  $s0, $s1, target
 addi $s1, $s1, 1
 sub  $s1, $s1, $s0
```

Is this "targeted" instruction executed or not? and its results?

```
target:        # label
 add  $s1, $s1, $s0
```

Labels indicate instruction location. They can't be reserved words and must be followed by colon (:)

Conditional Branching (beq)

```
# MIPS assembly
 addi $s0, $0, 4
 addi $s1, $0, 1
 sll  $s1, $s1, 2
 beq  $s0, $s1, target
 addi $s1, $s1, 1
 sub  $s1, $s1, $s0
```

```
# Results per line?
# $s0 = 0 + 4 = 4
# $s1 = 0 + 1 = 1
# $s1 = 1 << 2 = 4
# branch is taken
# not executed
# not executed
```

Is this "targeted" instruction executed or not? and its results?

```
target:        # label
 add  $s1, $s1, $s0
```

Labels indicate instruction location. They can't be reserved words and must be followed by colon (:)

Conditional Branching (beq)

```
# MIPS assembly                  Results per line?
 addi $s0, $0, 4                # $s0 = 0 + 4 = 4
 addi $s1, $0, 1                # $s1 = 0 + 1 = 1
 sll  $s1, $s1, 2               # $s1 = 1 << 2 = 4
 beq  $s0, $s1, target          # branch is taken
 addi $s1, $s1, 1               # not executed
 sub  $s1, $s1, $s0             # not executed
```

Is this "targeted" instruction executed or not? and its results?

```
target:          # label
 add  $s1, $s1, $s0      # $s1 = 4 + 4 = 8
```

Labels indicate instruction location. They can't be reserved words and must be followed by colon (:)

The Branch Not Taken (bne)

```
# MIPS assembly
   addi   $s0, $0, 4              # $s0 = 0 + 4 = 4
   addi   $s1, $0, 1              # $s1 = 0 + 1 = 1
   sll    $s1, $s1, 2             # $s1 = 1 << 2 = 4
   bne    $s0, $s1, target
   addi   $s1, $s1, 1
   sub    $s1, $s1, $s0


target:
   add    $s1, $s1, $s0
```

The Branch Not Taken (bne)

 # MIPS assembly

```
   addi   $s0, $0, 4          # $s0 = 0 + 4 = 4
   addi   $s1, $0, 1          # $s1 = 0 + 1 = 1
   sll    $s1, $s1, 2         # $s1 = 1 << 2 = 4
   bne    $s0, $s1, target
   addi   $s1, $s1, 1
   sub    $s1, $s1, $s0
```

Is this "targeted" instruction executed or not? and its results?

```
target:
   add    $s1, $s1, $s0
```

The Branch Not Taken (bne)

```
# MIPS assembly
   addi   $s0, $0, 4              # $s0 = 0 + 4 = 4
   addi   $s1, $0, 1              # $s1 = 0 + 1 = 1
   sll    $s1, $s1, 2             # $s1 = 1 << 2 = 4
   bne    $s0, $s1, target        # branch not taken
   addi   $s1, $s1, 1             # $s1 = 4 + 1 = 5
   sub    $s1, $s1, $s0           # $s1 = 5 - 4 = 1
```

Is this "targeted" instruction executed or not? and its results?

```
target:
   add    $s1, $s1, $s0
```

The Branch Not Taken (bne)

```
# MIPS assembly
   addi   $s0, $0, 4          # $s0 = 0 + 4 = 4
   addi   $s1, $0, 1          # $s1 = 0 + 1 = 1
   sll    $s1, $s1, 2         # $s1 = 1 << 2 = 4
   bne    $s0, $s1, target    # branch not taken
   addi   $s1, $s1, 1         # $s1 = 4 + 1 = 5
   sub    $s1, $s1, $s0       # $s1 = 5 - 4 = 1
```

Is this "targeted" instruction executed or not? and its results?

```
target:
   add    $s1, $s1, $s0    # dependent on where this line is
                           # usually, it is executed
```

Midterm2 Review

- Topics:
  - MIPS instruction basics
    - Pseudo-instruction to “real” MIPS instructions (non-pseudo)

MIPS Instructions: Arithmetic

Instruction	Example	Meaning	Comments
add	`add $1,$2,$3`	**$1=$2+$3**	
subtract	`sub $1,$2,$3`	**$1=$2-$3**	
add immediate	`addi $1,$2,100`	**$1=$2+100**	"Immediate" means a constant number
add unsigned	`addu $1,$2,$3`	**$1=$2+$3**	Values are treated as unsigned integers, not two's complement integers
subtract unsigned	`subu $1,$2,$3`	**$1=$2-$3**	Values are treated as unsigned integers, not two's complement integers

MIPS Instructions: Arithmetic

add immediate unsigned	`addiu $1,$2,100`	$1=$2+100	Values are treated as unsigned integers, not two's complement integers
Multiply (without overflow)	`mul $1,$2,$3`	$1=$2*$3	Result is only 32 bits!
Multiply	`mult $2,$3`	$hi,$low=$2*$3	Upper 32 bits stored in special register `hi` Lower 32 bits stored in special register `lo`
Divide	`div $2,$3`	$hi,$low=$2/$3	Remainder stored in special register `hi` Quotient stored in special register `lo`

MIPS Instructions: Logical

Instruction	Example	Meaning	Comments
and	`and $1,$2,$3`	$1=$2&$3	Bitwise AND
or	`or $1,$2,$3`	$1=$2|$3	Bitwise OR
and immediate	`andi $1,$2,100`	$1=$2&100	Bitwise AND with immediate value
or immediate	`or $1,$2,100`	$1=$2|100	Bitwise OR with immediate value
shift left logical	`sll $1,$2,10`	$1=$2<<10	Shift left by constant number of bits
shift right logical	`srl $1,$2,10`	$1=$2>>10	Shift right by constant number of bits

MIPS Instructions: Data Transfer

Instruction	Example	Meaning	Comments
load word	`lw $1,100($2)`	$1=Memory[$2+100]	Copy from memory to register
store word	`sw $1,100($2)`	Memory[$2+100]=$1	Copy from register to memory
load upper immediate	`lui $1,100`	$1=100x2^16	Load constant into upper 16 bits. Lower 16 bits are set to zero.
load address	`la $1,label`	$1=Address of label	*Pseudo-instruction* (provided by assembler, not processor!) Loads computed address of label (not its contents) into register
load immediate	`li $1,100`	$1=100	*Pseudo-instruction* (provided by assembler, not processor!) Loads immediate value into register

MIPS Instructions: Data Transfer

move from hi	`mfhi $2`	$2=hi	Copy from special register `hi` to general register
move from lo	`mflo $2`	$2=lo	Copy from special register `lo` to general register
move	`move $1,$2`	$1=$2	*Pseudo-instruction* (provided by assembler, not processor!) Copy from register to register.

Midterm2

- Topics:
  - For a given code snippet
    - Straightforward conversion

Less Than Comparison

- slt sets rd to 1 when rs < rt, otherwise, rd is 0

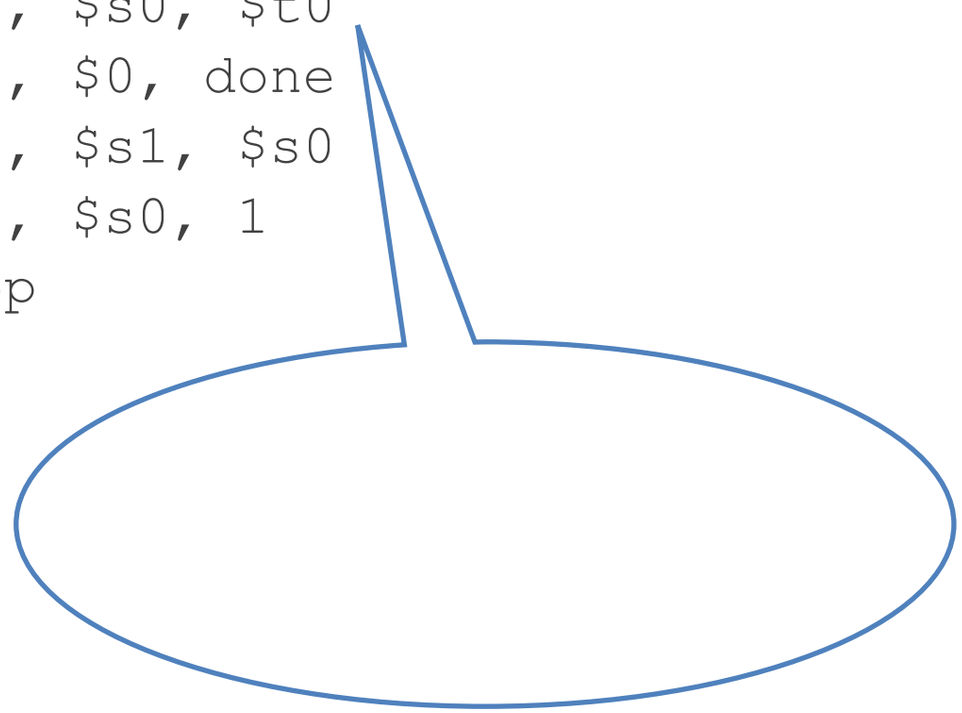
C Code

```
// add the powers of 2 from 1
// to 100
int sum = 0;
int i;

for (i=1; i < 101; i = i*2) {
   sum = sum + i;
}
```

MIPS assembly code

```
# $s0 = i, $s1 = sum
        addi $s1, $0, 0
        addi $s0, $0, 1
        addi $t0, $0, 101
loop:   slt  $t1, $s0, $t0
        beq  $t1, $0, done
        add  $s1, $s1, $s0
        sll  $s0, $s0, 1
        j    loop
done:
```



$t1 = 1 if i < 101

Less Than Comparison

- slt sets rd to 1 when rs < rt, otherwise, rd is 0

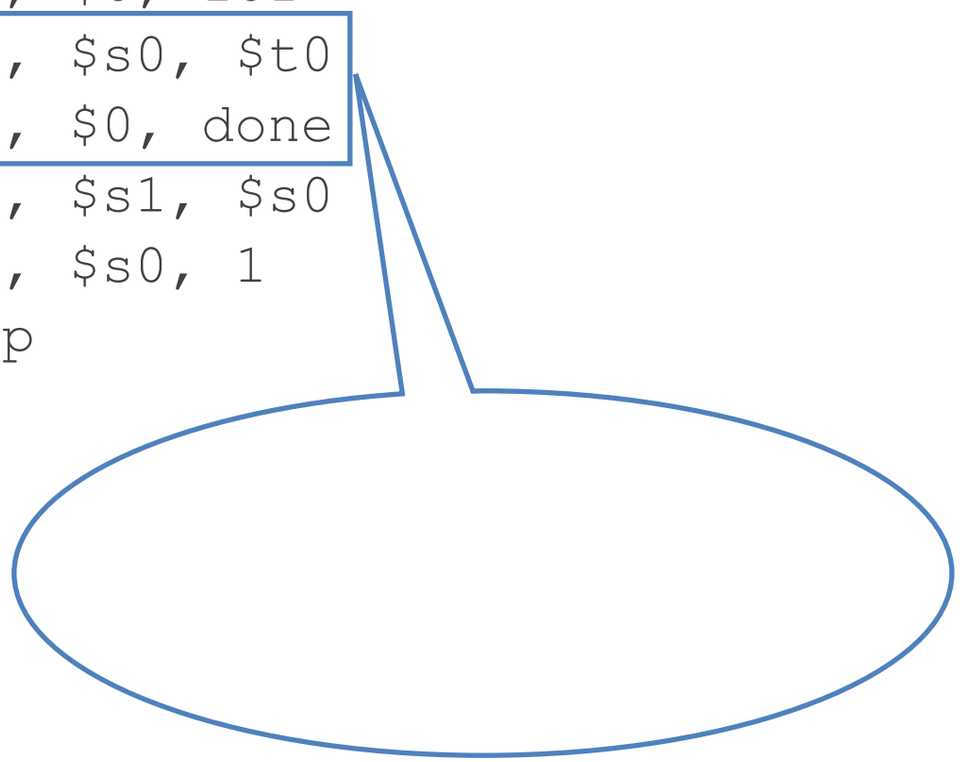
C Code

```
// add the powers of 2 from 1
// to 100
int sum = 0;
int i;

for (i=1; i < 101; i = i*2) {
   sum = sum + i;
}
```

MIPS assembly code

```
# $s0 = i, $s1 = sum
        addi $s1, $0, 0
        addi $s0, $0, 1
        addi $t0, $0, 101
loop:   slt  $t1, $s0, $t0
        beq  $t1, $0, done
        add  $s1, $s1, $s0
        sll  $s0, $s0, 1
        j    loop
done:
```



$t1 = 1 if i < 101

Less Than Comparison

- slt sets rd to 1 when rs < rt, otherwise, rd is 0

C Code

```
// add the powers of 2 from 1
// to 100
int sum = 0;
int i;

for (i=1; i < 101; i = i*2) {
   sum = sum + i;
}
```

MIPS assembly code

```
# $s0 = i, $s1 = sum
        addi $s1, $0, 0
        addi $s0, $0, 1
        addi $t0, $0, 101
loop:   slt  $t1, $s0, $t0
        beq  $t1, $0, done
        add  $s1, $s1, $s0
        sll  $s0, $s0, 1
        j    loop
done:
```

$t1 = 1 if i < 101

```
#if(i<101)$t1=1,else$t1=0
# if $t1 == 0 (i >= 101),
   branch to done
```

While Loops

C Code

```
// determines the power
// of x such that 2^x = 128
int pow = 1;
int x   = 0;

while (pow != 128) {
  pow = pow * 2;
  x = x + 1;
}
```

MIPS assembly code

```
# $s0 = pow, $s1 = x

        addi $s0, $0, 1
        add  $s1, $0, $0
        addi $t0, $0, 128
while:  beq  $s0, $t0, done
        sll  $s0, $s0, 1
        addi $s1, $s1, 1
        j    while
done:
```

Assembly tests for the opposite case (`pow == 128`) of the C code (`pow != 128`).

For Loops

C Code

```
// add the numbers from 0 to 9
int sum = 0;
int i;

for (i=0; i!=10; i = i+1) {
  sum = sum + i;
}
```

MIPS assembly code

```
# $s0 = i, $s1 = sum
        addi $s1, $0, 0
        add  $s0, $0, $0
        addi $t0, $0, 10
for:    beq  $s0, $t0, done
        add  $s1, $s1, $s0
        addi $s0, $s0, 1
        j    for
done:
```

while and for side-by-side

MIPS assembly code

```
# $s0 = pow, $s1 = x

        addi $s0, $0, 1
        add  $s1, $0, $0
        addi $t0, $0, 128
while:  beq  $s0, $t0, done
        sll  $s0, $s0, 1
        addi $s1, $s1, 1
        j    while
done:
```

MIPS assembly code

```
# $s0 = i, $s1 = sum
        addi $s1, $0, 0
        add  $s0, $0, $0
        addi $t0, $0, 10
for:    beq  $s0, $t0, done
        add  $s1, $s1, $s0
        addi $s0, $s0, 1
        j    for
done:
```

Difference/Similarity between for and while Loop

- From logic, they are doing "exactly" the same thing
- If being used to solve the same problem, the assembly code will be the same

- Difference, more from the programming level
- Using the for loop, you usually know the number of iterations
- Using the while loop, you just need to know when/how to finish the loop

Midterm2 Review

- Topics:
  - For a given code snippet
    - Caller and Calle involved (stack!)

Function Calls

- **Caller:** calling function (in this case, `main`)
- **Callee:** called function (in this case, `sum`)

C Code

```
void main()
{
  int y;
  y = sum(42, 7);
  ...
}

int sum(int a, int b)
{
  return (a + b);
}
```

Function Conventions

- **Caller:**
  - passes **arguments** to callee
  - jumps to callee
- **Callee:**
  - **performs** the function
  - **returns** result to caller
  - **returns** to point of call
- **must not overwrite** registers or memory needed by caller

C Code

```
void main()
{
  int y;
  y = sum(42, 7);
  ...
}

int sum(int a, int b)
{
  return (a + b);
}
```

MIPS Function Conventions

- **Call Function:** jump and link (`jal`)
- **Return** from function: jump register (`jr`)

Function Calls

C Code

```
int main() {
  simple();
  a = b + c;
}

void simple() {
  return;
}
```

MIPS assembly code

```
0x00400200 main: jal  simple
0x00400204       add  $s0, $s1, $s2
...

0x00401020 simple: jr $ra
```

void —> simple doesn't return a value

Function Calls

C Code

```
int main() {
  simple();
  a = b + c;
}

void simple() {
  return;
}
```

MIPS assembly code

```
0x00400200 main: jal  simple
0x00400204       add  $s0, $s1, $s2
...

0x00401020 simple: jr $ra
```

jal: jumps to `simple`
`$ra` = PC + 4 = 0x00400204

jr $ra: jumps to address in `$ra` (0x00400204)

Revisit the MIPS Register Set

Name	Register Number	Usage
$0	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2-3	Function return values
$a0-$a3	4-7	Function arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved variables
$t8-$t9	24-25	more temporaries
$k0-$k1	26-27	OS temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	Function return address

Revisit the MIPS Register Set

Name	Register Number	Usage
$0	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2-3	Function return values
$a0-$a3	4-7	Function arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved variables
$t8-$t9	24-25	more temporaries
$k0-$k1	26-27	OS temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	Function return address

Revisit the MIPS Register Set

Name	Register Number	Usage
$0	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2-3	Function return values
$a0-$a3	4-7	Function arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved variables
$t8-$t9	24-25	more temporaries
$k0-$k1	26-27	OS temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	Function return address

Input Arguments & Return Value

MIPS conventions:

- Argument values: `$a0 - $a3`
- Return values: `$v0-$v1`

Input Arguments & Return Value

C Code

```
int main()
{
  int y;
  ...
  y = diffofsums(2, 3, 4, 5);  // 4 arguments
  ...
}

int diffofsums(int f, int g, int h, int i)
{
  int result;
  result = (f + g) - (h + i);
  return result;                // return value
}
```

Input Arguments & Return Value

MIPS assembly code

```
# $s0 = y

main:
   ...
   addi $a0, $0, 2     # argument 0 = 2
   addi $a1, $0, 3     # argument 1 = 3
   addi $a2, $0, 4     # argument 2 = 4
   addi $a3, $0, 5     # argument 3 = 5
   jal  diffofsums     # call Function
   add  $s0, $v0, $0   # y = returned value
   ...

# $s0 = result
diffofsums:
   add $t0, $a0, $a1   # $t0 = f + g
   add $t1, $a2, $a3   # $t1 = h + i
   sub $s0, $t0, $t1   # result = (f + g) - (h + i)
   add $v0, $s0, $0    # put return value in $v0
   jr  $ra             # return to caller
```

Input Arguments & Return Value

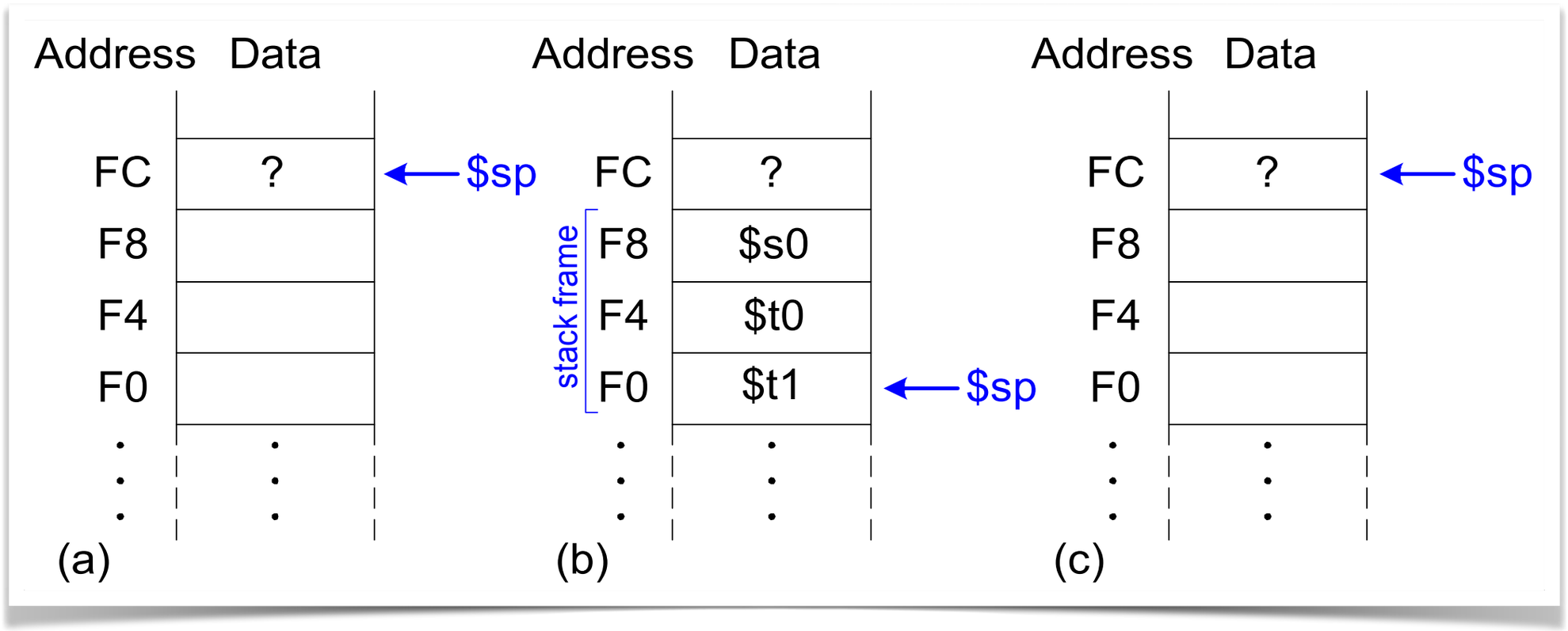
MIPS assembly code

```
# $s0 = result
diffofsums:
   add $t0, $a0, $a1   # $t0 = f + g
   add $t1, $a2, $a3   # $t1 = h + i
   sub $s0, $t0, $t1   # result = (f + g) - (h + i)
   add $v0, $s0, $0    # put return value in $v0
   jr  $ra             # return to caller
```

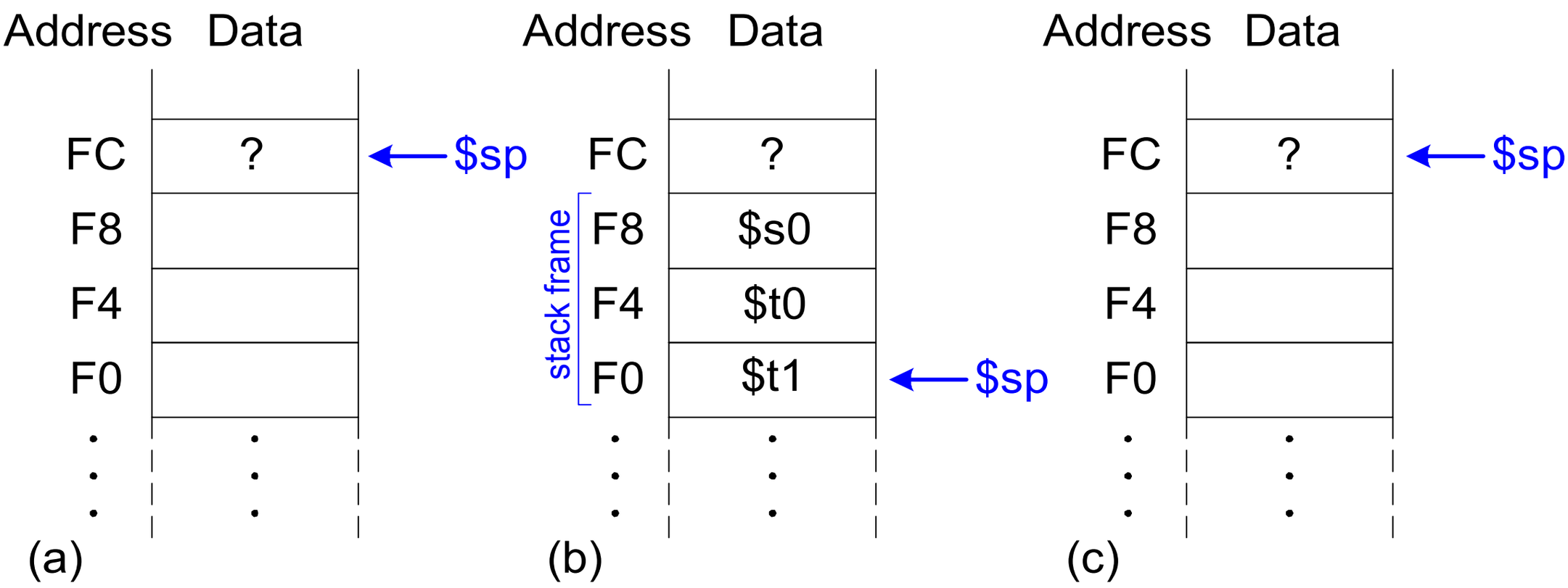
- `diffofsums` overwrite 3 registers: `$t0`, `$t1`, `$s0`
- `diffofsums` can use *stack* to temporarily store registers

Storing Register Values on the Stack

```
# $s0 = result
diffofsums:
  addi $sp, $sp, -12  # make space on stack
                      # to store 3 registers
  sw   $s0, 8($sp)    # save $s0 on stack
  sw   $t0, 4($sp)    # save $t0 on stack
  sw   $t1, 0($sp)    # save $t1 on stack
  add  $t0, $a0, $a1  # $t0 = f + g
  .................
```

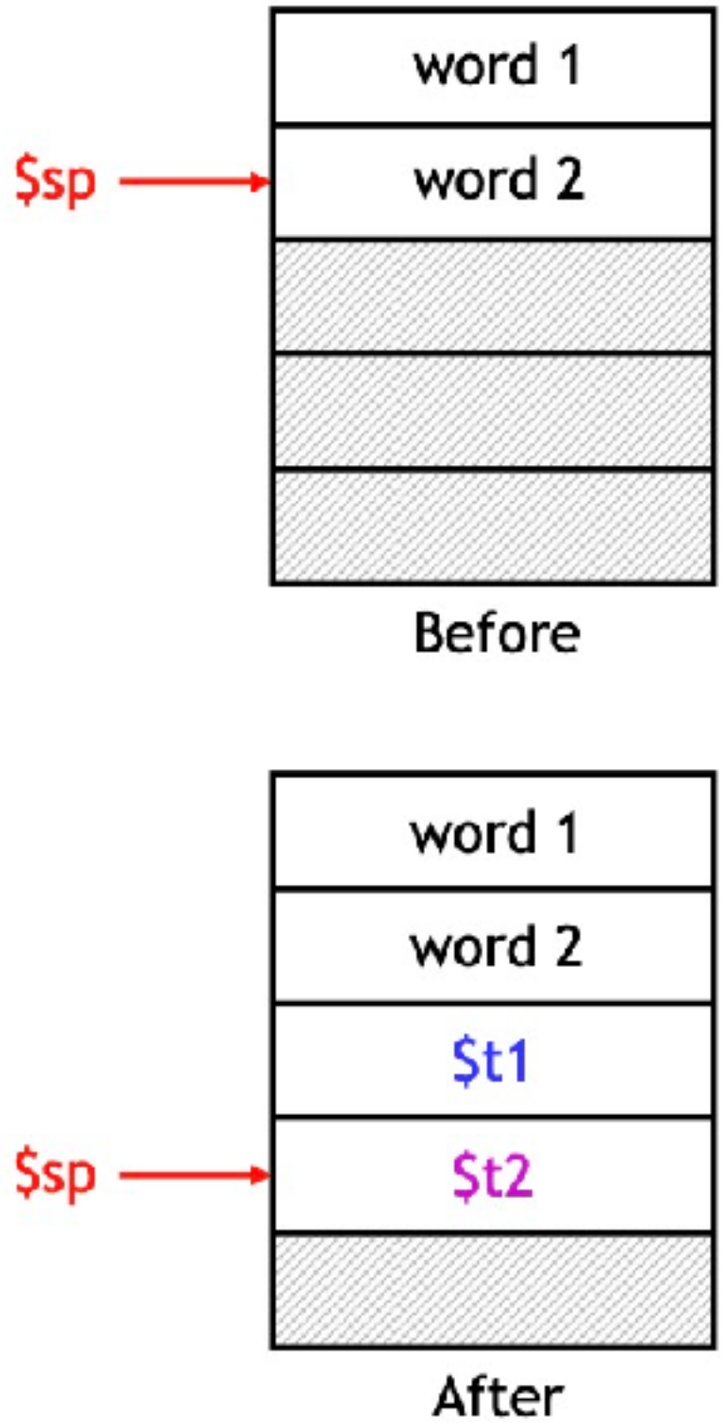


The stack during diffofsums Call



"Pushing" Elements

- To push elements onto the stack:
  - Move the stack pointer $sp down to make room for the new data.
  - Store the elements into the stack.

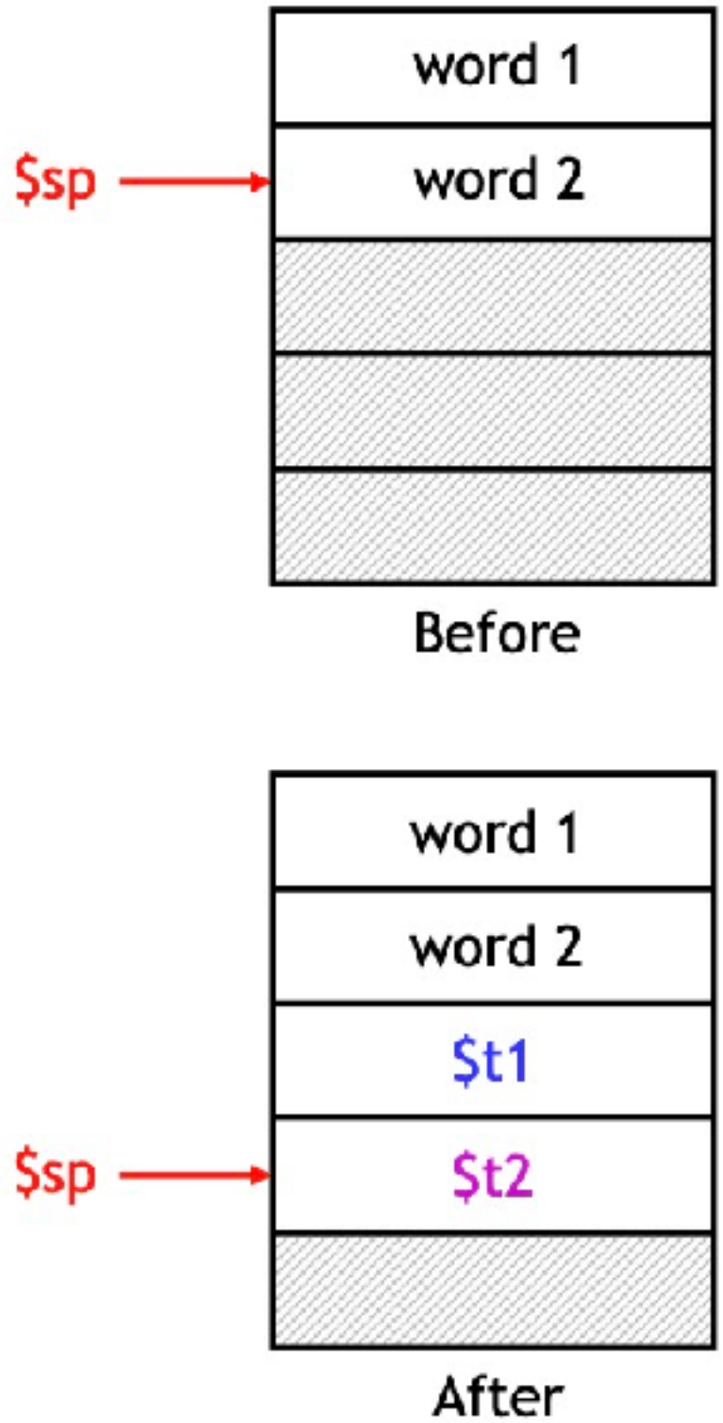


- For example, to push registers $t1 and $t2 onto the stack:

```
addi $sp, $sp, -8
sw   $t1, 4($sp)
sw   $t2, 0($sp)
```

"Pushing" Elements

- To push elements onto the stack:
  - Move the stack pointer $sp down to make room for the new data.
  - Store the elements into the stack.

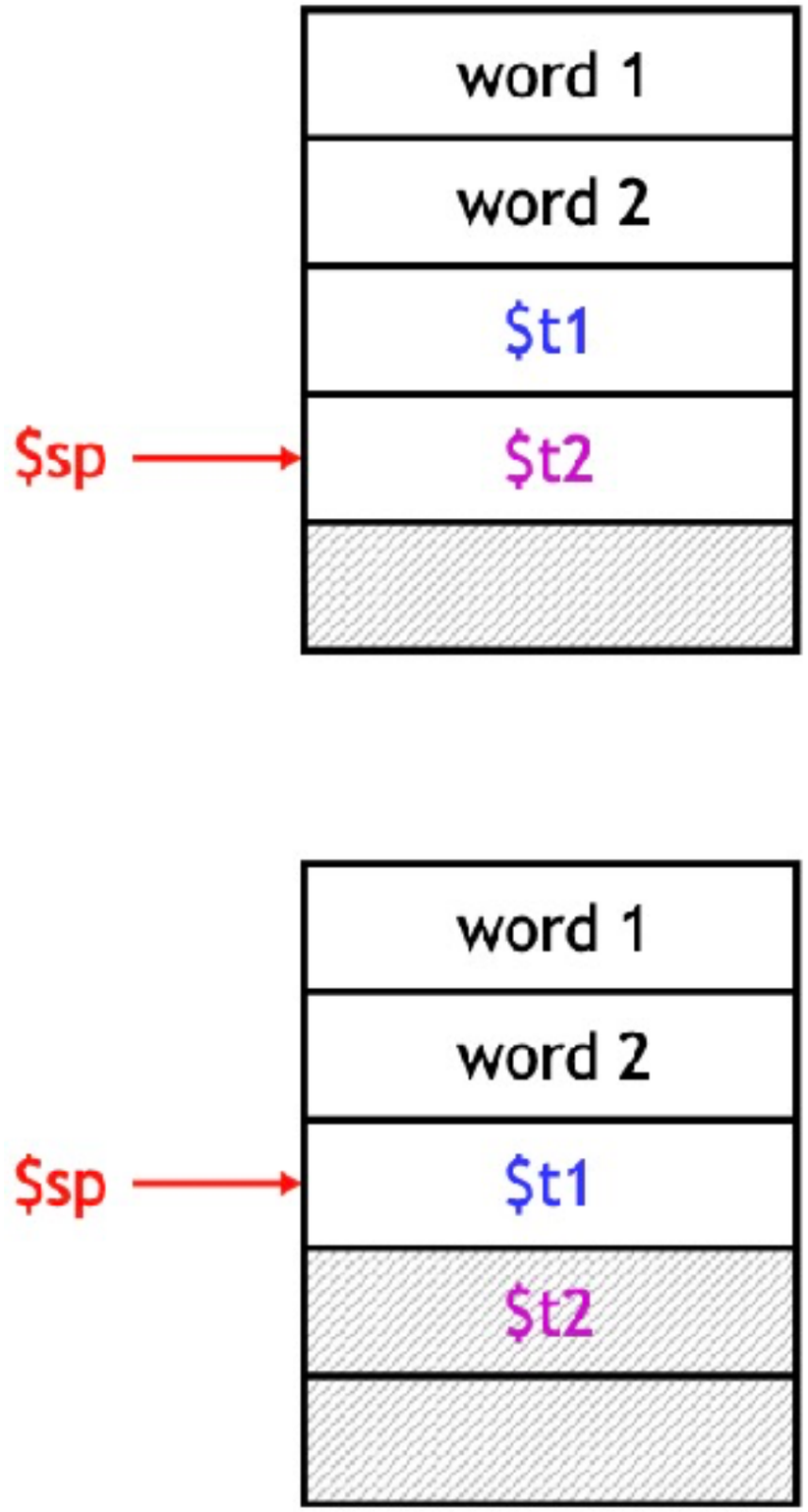


- Equivalent stack operations:

```
sw   $t1, -4($sp)
sw   $t2, -8($sp)
addi $sp, $sp, -8
```

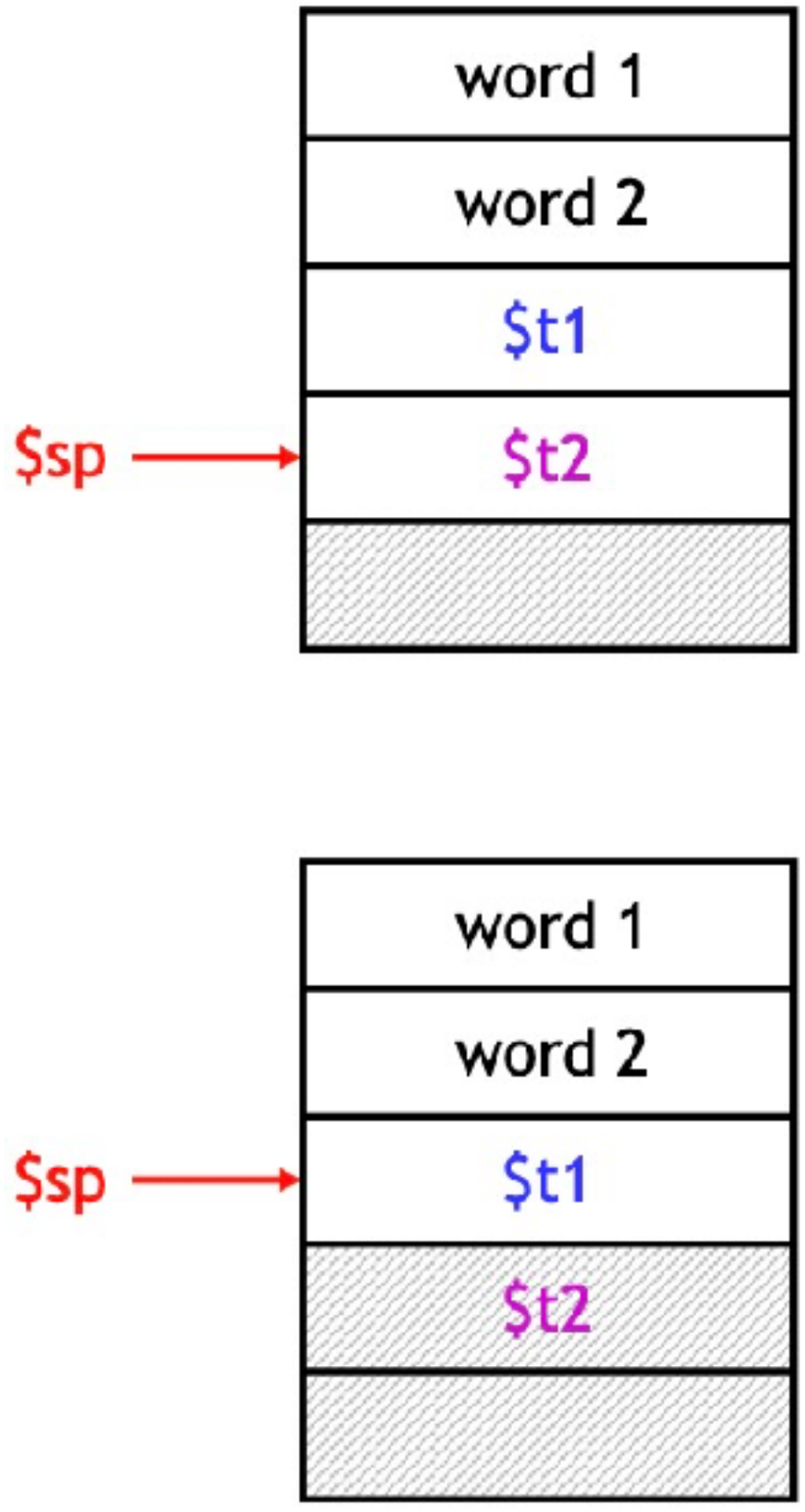
Accessing Elements in Stack

- A user can access to any stack element, Not just the top one —> pointed by $sp!
- But, s/he should know the location, i.e., relative to $sp!
- For example, to retrieve the value of $t1:
  - **lw $s0, 4($sp)**



Accessing Elements in Stack

- To pop the value of $t2, yielding the stack shown at the bottom:
  - **addi $sp, $sp, 4**
  - **Note that $t2 is still valid in the memory, if you want to erase it, reset the stack point!**



Storing Register Values on the Stack

```
# $s0 = result
diffofsums:
  addi $sp, $sp, -12  # make space on stack
                      # to store 3 registers
  sw   $s0, 8($sp)    # save $s0 on stack
  sw   $t0, 4($sp)    # save $t0 on stack
  sw   $t1, 0($sp)    # save $t1 on stack
  add  $t0, $a0, $a1  # $t0 = f + g
  add  $t1, $a2, $a3  # $t1 = h + i
  sub  $s0, $t0, $t1  # result = (f + g) - (h + i)
  add  $v0, $s0, $0   # put return value in $v0
  lw   $t1, 0($sp)    # restore $t1 from stack
  lw   $t0, 4($sp)    # restore $t0 from stack
  lw   $s0, 8($sp)    # restore $s0 from stack
  addi $sp, $sp, 12   # deallocate stack space
  jr   $ra            # return to caller
```

Storing Register Values on the Stack

```
# $s0 = result
diffofsums:
  addi $sp, $sp, -12   # make space on stack
                       # to store 3 registers
  sw   $s0, 8($sp)     # save $s0 on stack
  sw   $t0, 4($sp)     # save $t0 on stack
  sw   $t1, 0($sp)     # save $t1 on stack
  add  $t0, $a0, $a1   # $t0 = f + g
  add  $t1, $a2, $a3   # $t1 = h + i
  sub  $s0, $t0, $t1   # result = (f + g) - (h + i)
  add  $v0, $s0, $0    # put return value in $v0
  lw   $t1, 0($sp)     # restore $t1 from stack
  lw   $t0, 4($sp)     # restore $t0 from stack
  lw   $s0, 8($sp)     # restore $s0 from stack
  addi $sp, $sp, 12    # deallocate stack space
  jr   $ra             # return to caller
```

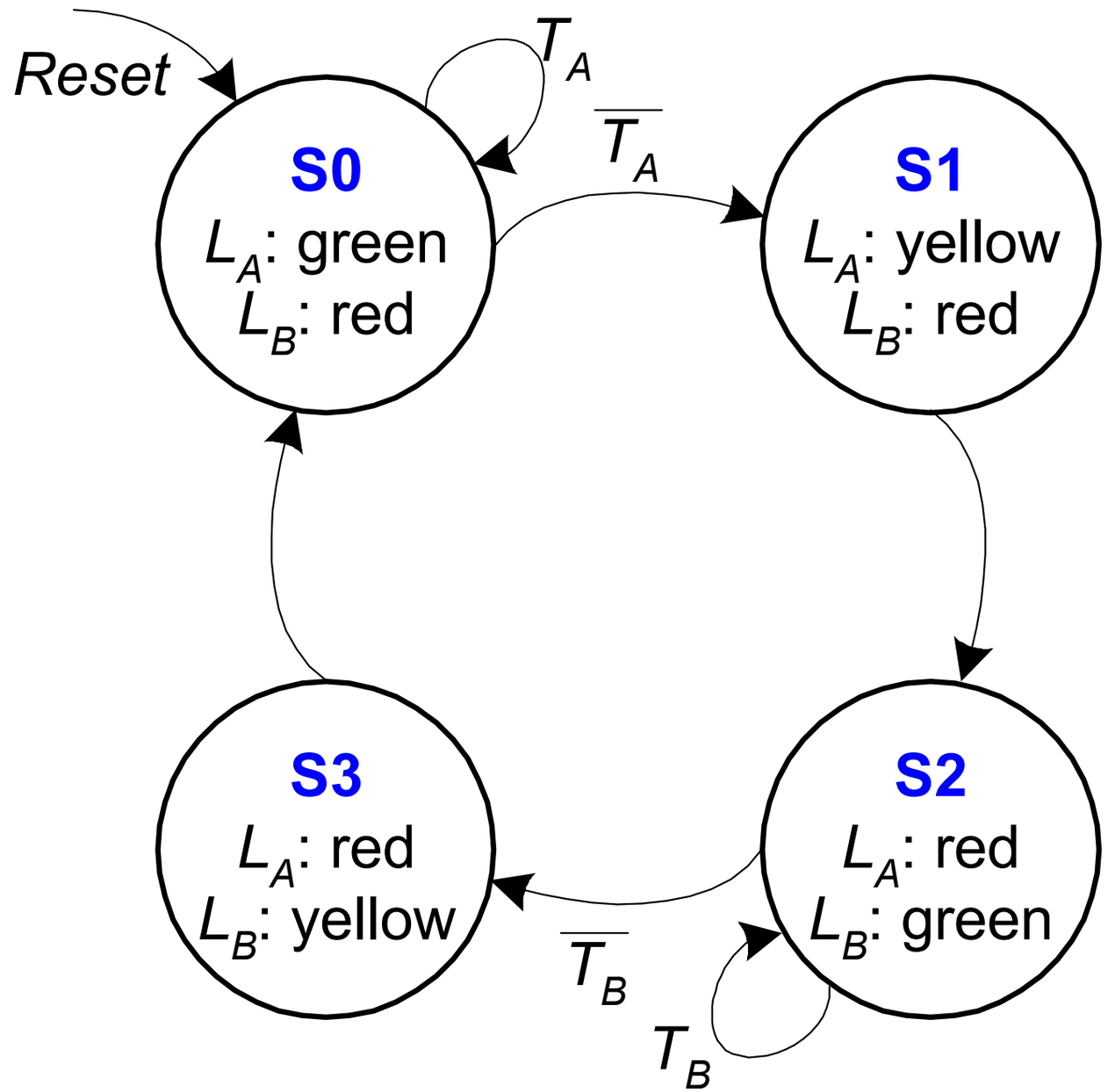
- **Doable, but too complicated and low-efficiency!**

Midterm2 Review

- Topics:
  - FSM, analysis and design

FSM State Transition Diagram

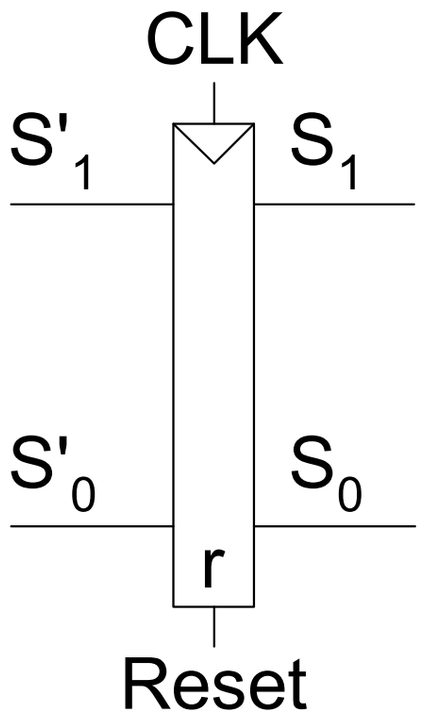
- **Moore FSM:** outputs labeled in each state
- **States:** Circles
- **Transitions:** Arcs



FSM Schematic: State Register

$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\overline{S_0}\,\overline{T_B}$$

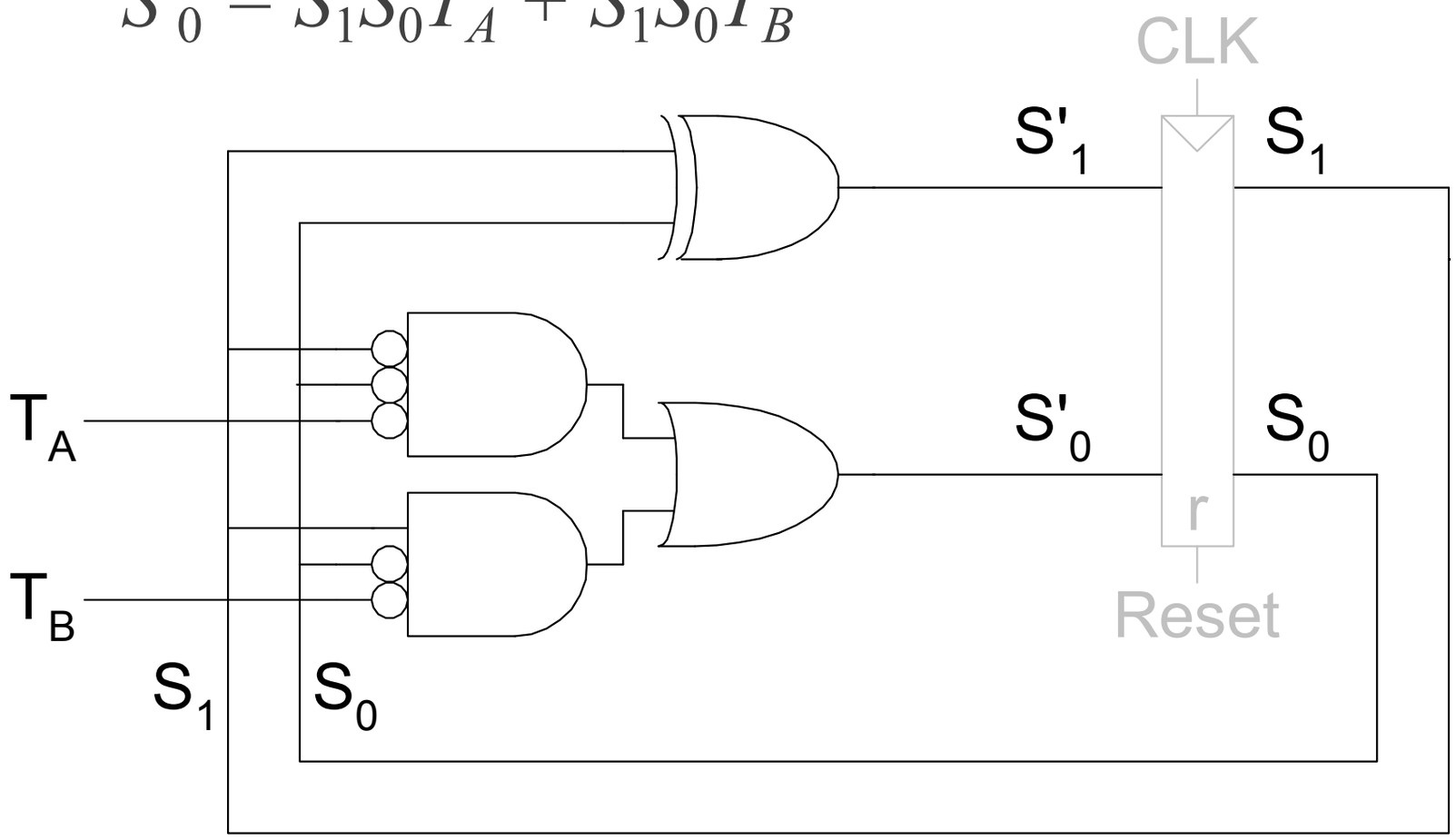


state register

FSM Schematic: Next State Logic

$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = S_1 S_0 T_A + S_1 S_0 T_B$$



$$L_{A1} = S_1$$

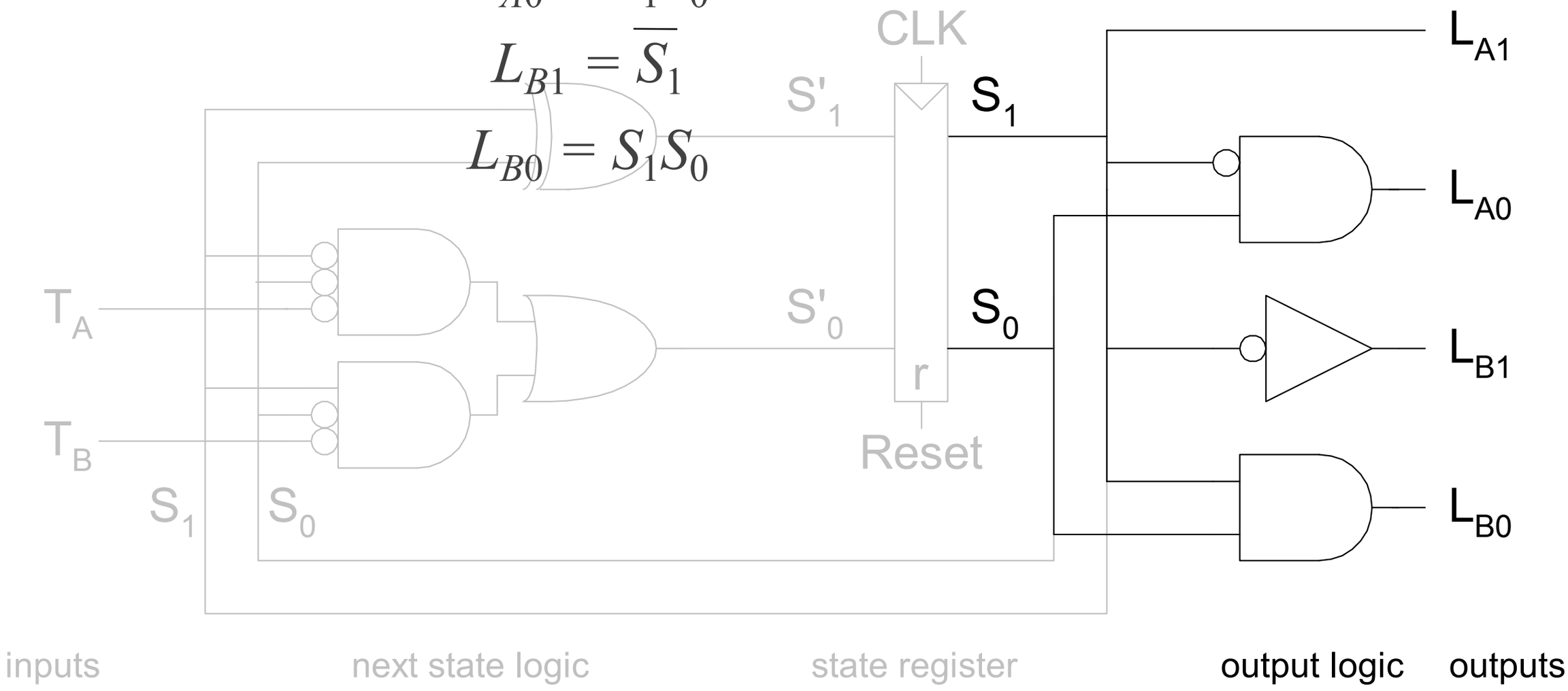
$$L_{A0} = \overline{S_1} S_0$$

$$L_{B1} = \overline{S_1}$$

$$L_{B0} = S_1 S_0$$

FSM Schematic: Output Logic

$$L_{A1} = S_1$$
$$L_{A0} = \overline{S_1} S_0$$
$$L_{B1} = \overline{S_1}$$
$$L_{B0} = S_1 S_0$$



FSM Design Procedure

- Identify inputs and outputs
- Sketch state transition diagram
- Write state transition table
- Select state encodings
- For Moore machine:
  - Rewrite state transition table with state encodings
  - Write output table

FSM Design Procedure

- Identify inputs and outputs
- Sketch state transition diagram
- Write state transition table
- Select state encodings
- For a Mealy machine:
  - Rewrite combined state transition and output table with state encodings

Last Step

- Write Boolean equations for next state and output logic
- Sketch the circuit schematic

A Design Example

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

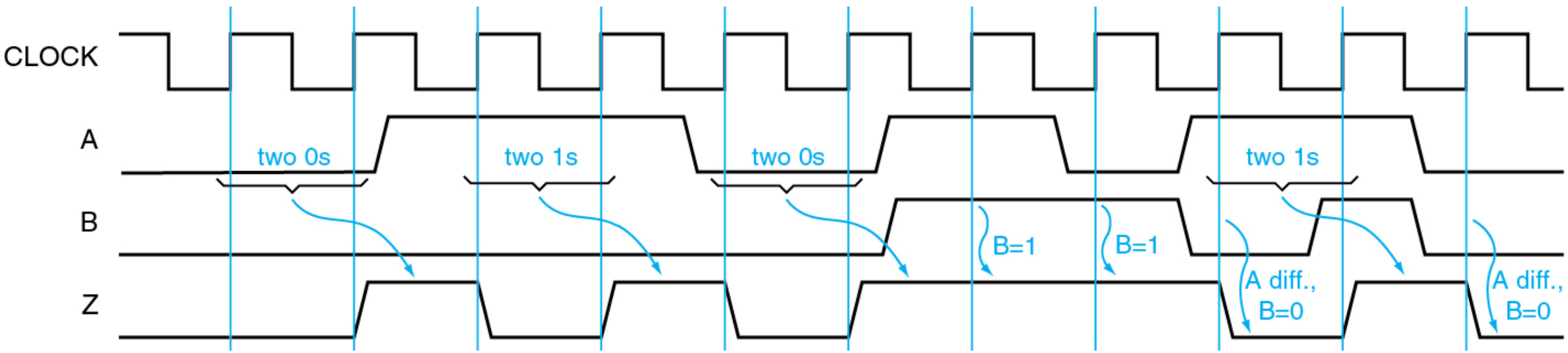
Otherwise, the output should be 0.

A Design Example

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.



A Design Example: Review

(a)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT					0
	. . .					
	. . .					
	. . .					
		S				

(b)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0					0
Got a 1 on A	A1					0
		S				

(c)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1					0
Got two equal A inputs	OK					1
		S				

(d)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK					1
		S				

A Design Example: Review

(a)

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT					0
	. . .					
	. . .					
	. . .					

S

(b)

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0					0
Got a 1 on A	A1					0

S

(c)

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1					0
Got two equal A inputs	OK					1

S

(d)

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK					1

S

A Design Example: Review

(a)

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT					0
	. . .					
	. . .					
	. . .					
		S				

(b)

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0					0
Got a 1 on A	A1					0
		S				

(c)

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1					0
Got two equal A inputs	OK					1
		S				

(d)

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK					1
		S				

A Design Example: Review

(a)

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT					0
	. . .					
	. . .					
	. . .					

S

(b)

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0					0
Got a 1 on A	A1					0

S

(c)

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1					0
Got two equal A inputs	OK					1

S

(d)

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK					1

S

A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

(a)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK	?	OK	OK	?	1
		S				

(b)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0					1
Two equal, A=1 last	OK1					1
		S				

(c)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1					1
		S				

(d)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1	A0	OK0	OK1	OK1	1
		S				

A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

(a)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK	?	OK	OK	?	1
		S				

(b)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0					1
Two equal, A=1 last	OK1					1
		S				

(c)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1					1
		S				

(d)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1	A0	OK0	OK1	OK1	1
		S				

A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

(a)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK	?	OK	OK	?	1

S

(b)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0					1
Two equal, A=1 last	OK1					1

S

(c)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1					1

S

(d)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1	A0	OK0	OK1	OK1	1

S

A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

(a)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK	?	OK	OK	?	1

Z=1 triggered by B or A?

(b)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0					1
Two equal, A=1 last	OK1					1

(c)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1					1

(d)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1	A0	OK0	OK1	OK1	1

A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

(a)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK	?	OK	OK	?	1

Z=1 triggered by B or A?

(b)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0					1
Two equal, A=1 last	OK1					1

S

(c)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1					1

S

(d)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1	A0	OK0	OK1	OK1	1

S

A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

(a)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK	?	OK	OK	?	1

Z=1 triggered by B or A?

(b)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0					1
Two equal, A=1 last	OK1					1

S

(c)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1					1

S

(d)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1	A0	OK0	OK1	OK1	1

S

A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

(a)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK	?	OK	OK	?	1

Z=1 triggered by B or A?

(b)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0					1
Two equal, A=1 last	OK1					1
		S				

(c)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1					1
		S				

(d)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1	A0	OK0	OK1	OK1	1
		S				

EECE 2322: Fundamentals of Digital Design and Computer Organization

Lecture 11_2: FSM

Xiaolin Xu
Department of ECE
Northeastern University

A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

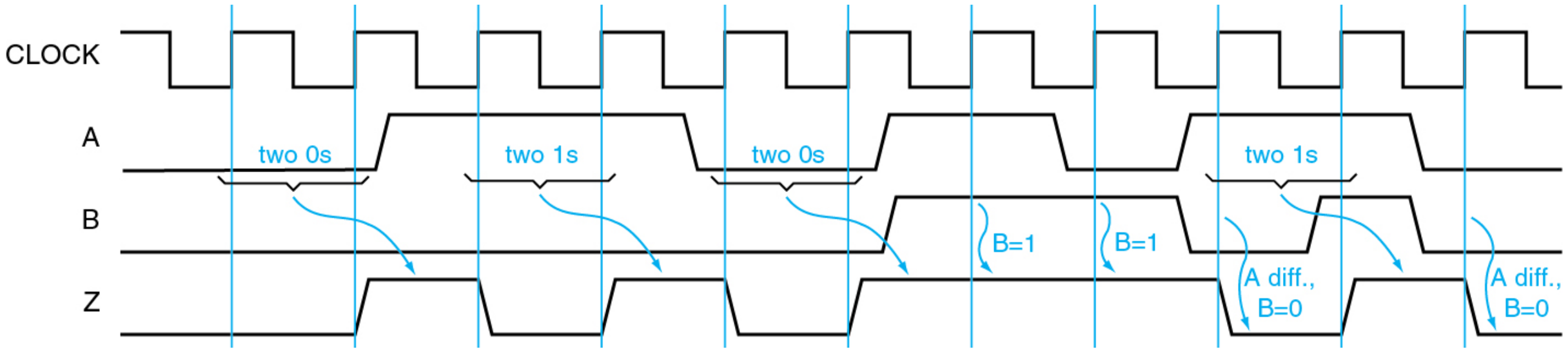
Otherwise, the output should be 0.

A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

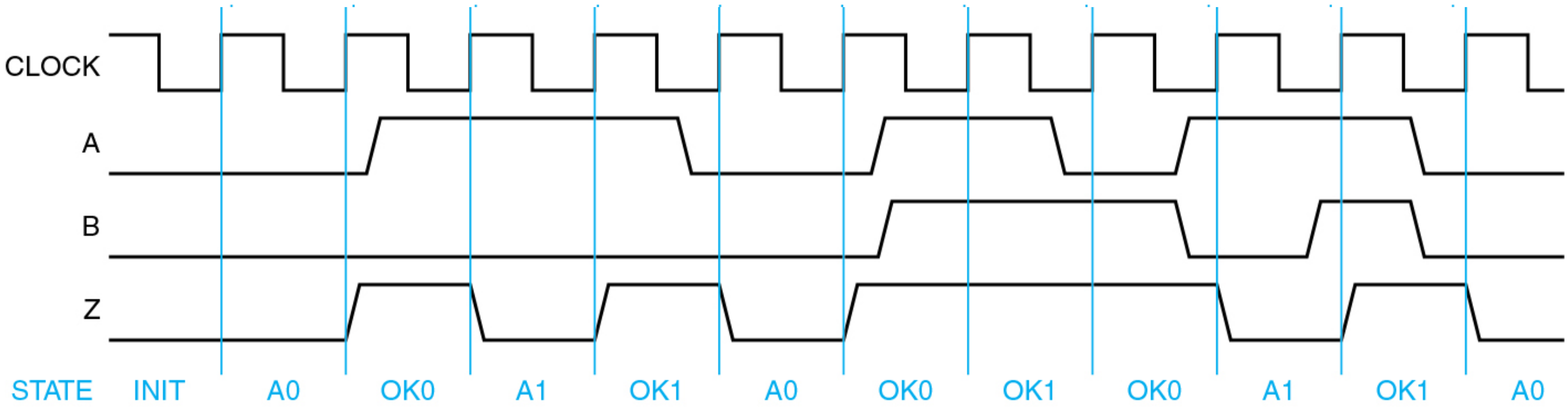


A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

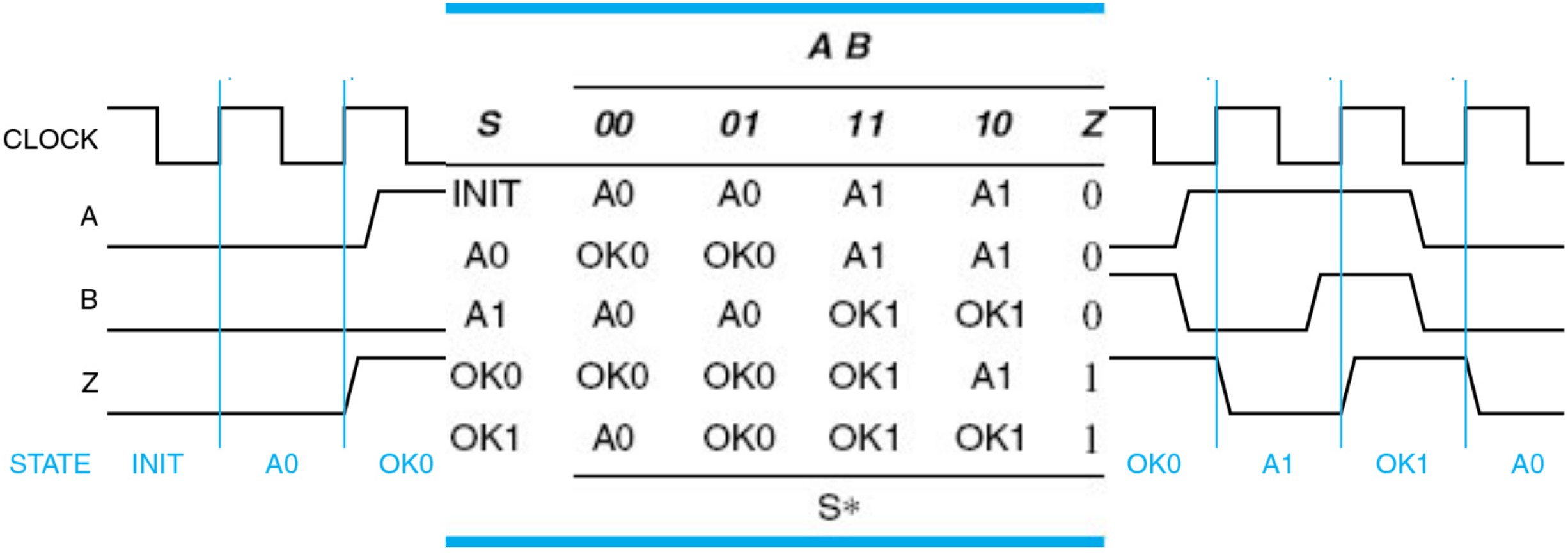


A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.



S	A B = 00	01	11	10	Z
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
	S*				

A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

	A B				
S	00	01	11	10	Z
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
	S*				

A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

	A B				
S	00	01	11	10	Z
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
	S*				

	Assignment			
State Name	Simplest Q1–Q3	Decomposed Q1–Q3	One-Hot Q1–Q5	Almost One-Hot Q1–Q4
INIT	000	000	00001	0000
A0	001	100	00010	0001
A1	010	101	00100	0010
OK0	011	110	01000	0100
OK1	100	111	10000	1000

A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

	A B				
S	00	01	11	10	Z
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
	S*				

A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

	A B				
S	**00**	**01**	**11**	**10**	**Z**
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
	S*				

	Assignment			
State Name	**Simplest Q1–Q3**	**Decomposed Q1–Q3**	**One-Hot Q1–Q5**	**Almost One-Hot Q1–Q4**
INIT	000	000	00001	0000
A0	001	100	00010	0001
A1	010	101	00100	0010
OK0	011	110	01000	0100
OK1	100	111	10000	1000

A Design Example: Review

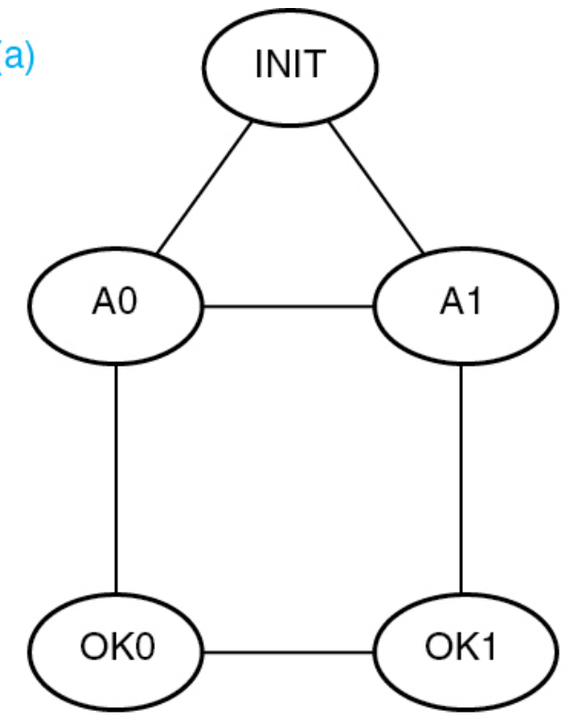
Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

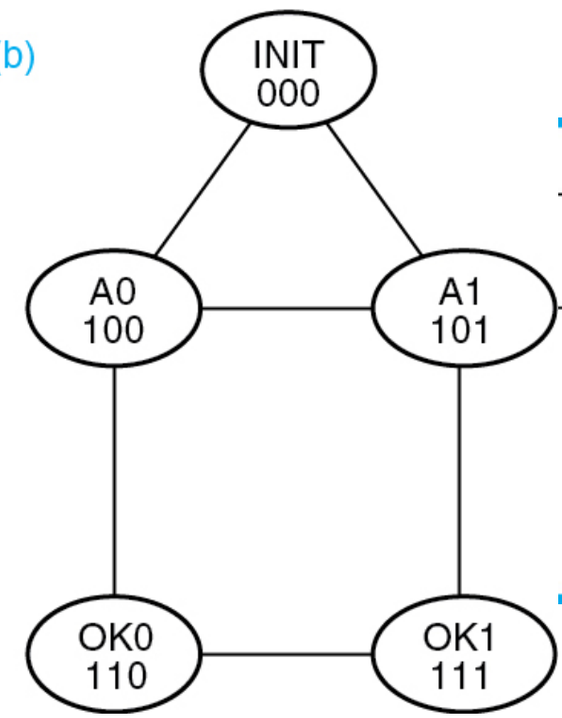
Otherwise, the output should be 0.

	A B				
S	00	01	11	10	Z
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
	S*				

(a)



(b)



A Design Example: Review

Design a clocked synchronous state machine with two inputs, A and B, and a single output Z that is 1 if:

- A had the same value at each of the two previous clock ticks, *or*
- B has been 1 since the last time that the first condition was true.

Otherwise, the output should be 0.

(a)

	A B				
S	00	01	11	10	Z
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
	S*				

	A B				
Q1 Q2 Q3	00	01	11	10	Z
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	Q1* Q2* Q3* or D1 D2 D3				

Verilog Coding for the Example

```
module VrSMex( CLOCK, A, B, Z );
  input CLOCK, A, B;
  output Z;
  reg Z;
  reg [2:0] Sreg, Snext;          // State register and next state
  parameter [2:0] INIT = 3'b000,  // Define the states
                  A0   = 3'b001,
                  A1   = 3'b010,
                  OK0  = 3'b011,
                  OK1  = 3'b100;
```

Verilog Coding for the Example

```
always @ (posedge CLOCK)  // Create the state memory
  Sreg <= Snext;

always @ (A, B, Sreg) begin  // Next-state logic
  case (Sreg)
    INIT:   if (A==0)  Snext = A0;
            else       Snext = A1;
    A0:     if (A==0)  Snext = OK0;
            else       Snext = A1;
    A1:     if (A==0)  Snext = A0;
            else       Snext = OK1;
    OK0:    if (A==0)  Snext = OK0;
            else if ((A==1) && (B==0)) Snext = A1;
            else                       Snext = OK1;
    OK1:    if ((A==0) && (B==0))      Snext = A0;
            else if ((A==0) && (B==1)) Snext = OK0;
            else                       Snext = OK1;
    default Snext = INIT;
  endcase
end
```

Verilog Coding for the Example

```
always @ (Sreg)                    // Output logic
  case (Sreg)
    INIT, A0, A1: Z = 0;
    OK0, OK1:     Z = 1;
    default       Z = 0;
  endcase
endmodule
```

Verilog Coding for the Example

```
always @ (Sreg)                    // Output logic
  case (Sreg)
    INIT, A0, A1: Z = 0;
    OK0, OK1:     Z = 1;
    default       Z = 0;
  endcase
endmodule
```