Midterm Review

CIS 2400 Recitation 6

Handouts

Access at

Review Topics

Vote for the topic you most want to cover! We'll go through them starting from the one with the most votes.

- Combinational logic
- Sequential logic
- Control signal circuit
- Decoding control signals
- Converting between LC4 ASM to binary instructions
- Writing LC4 code
- LC4 program trace

Combinational Logic 1

Given a 4-bit input $I_3I_2I_1I_0$, we want to know whether there are exactly 3 bits that are 1. To solve this problem:

1) Write out a boolean equation
2) Design a transistor-level circuit
3) Design a gate-level circuit

You may assume that you have access to inverted inputs.

Combinational Logic 1 – Equation

Equation should return 1 when the input is

- 1110
- 1101
- 1011
- 0111

$(I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, \sim I_0) \mid (I_3 \,\&\, I_2 \,\&\, \sim I_1 \,\&\, I_0) \mid (I_3 \,\&\, \sim I_2 \,\&\, I_1 \,\&\, I_0) \mid (\sim I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, I_0)$

Combinational Logic 1 – CMOS

PDN equation:

$$\sim((I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, \sim I_0) \mid (I_3 \,\&\, I_2 \,\&\, \sim I_1 \,\&\, I_0) \mid (I_3 \,\&\, \sim I_2 \,\&\, I_1 \,\&\, I_0) \mid (\sim I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, I_0))$$

$$= \sim(I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, \sim I_0) \,\&\, \sim(I_3 \,\&\, I_2 \,\&\, \sim I_1 \,\&\, I_0) \,\&\, \sim(I_3 \,\&\, \sim I_2 \,\&\, I_1 \,\&\, I_0) \,\&\, \sim(\sim I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, I_0)$$

$$= (\sim I_3 \mid \sim I_2 \mid \sim I_1 \mid I_0) \,\&\, (\sim I_3 \mid \sim I_2 \mid I_1 \mid \sim I_0) \,\&\, (\sim I_3 \mid I_2 \mid \sim I_1 \mid \sim I_0) \,\&\, (I_3 \mid \sim I_2 \mid \sim I_1 \mid \sim I_0)$$

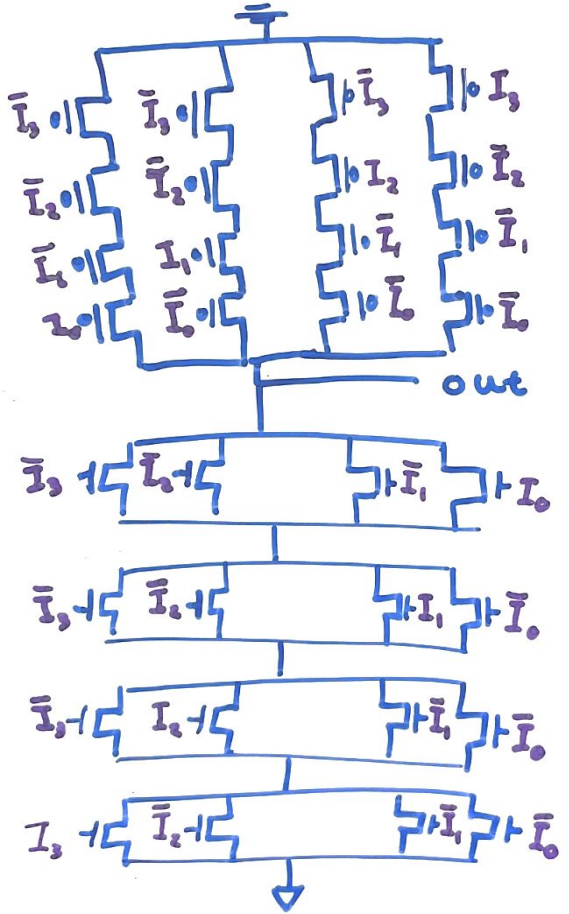
Combinational Logic 1 – CMOS

PDN equation:

$(\sim I_3 \mid \sim I_2 \mid \sim I_1 \mid I_0)$ & $(\sim I_3 \mid \sim I_2 \mid I_1 \mid \sim I_0)$ &

$(\sim I_3 \mid I_2 \mid \sim I_1 \mid \sim I_0)$ & $(I_3 \mid \sim I_2 \mid \sim I_1 \mid \sim I_0)$

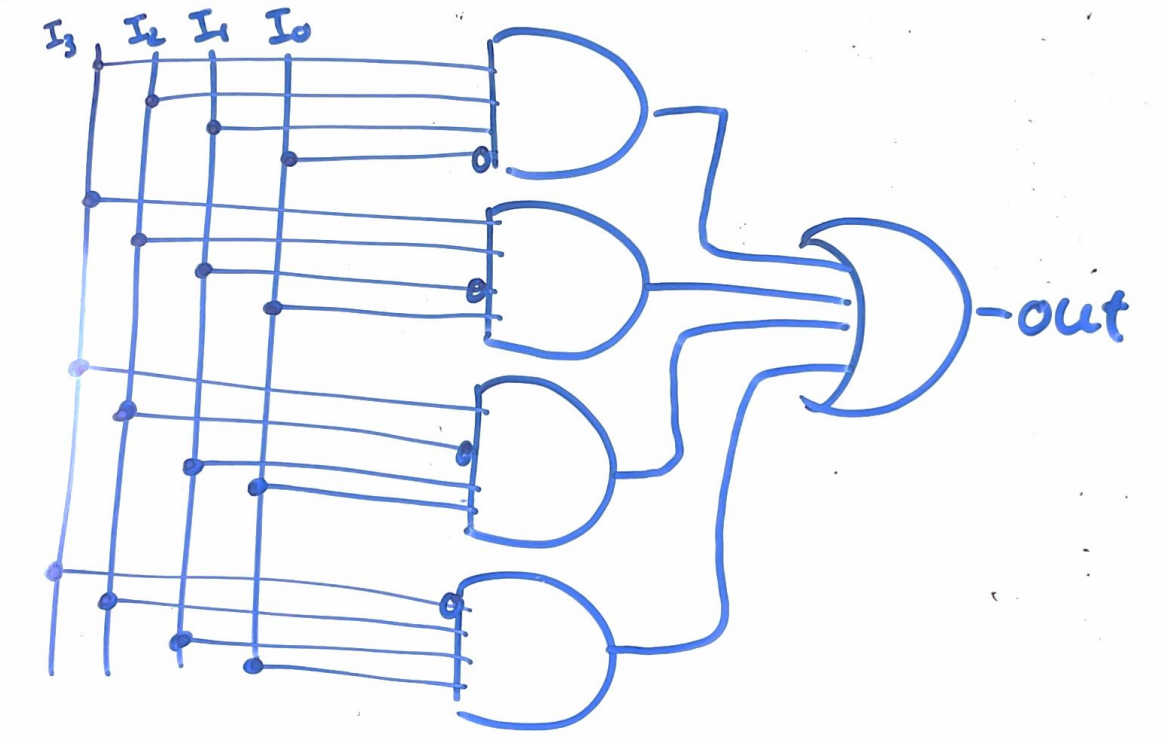
PUN is the structural complement of the PDN



Combinational Logic 1 – Gate Level

Option 1: PLA

Option 2: anything logically equivalent!



Combinational Logic 2

Given a 4-bit input X = $I_3 I_2 I_1 I_0$, we want to know whether X is divisible by 6. To solve this problem:

1) Write out a boolean equation
2) Design a transistor-level circuit
3) Design a gate-level circuit

You may assume that you have access to inverted inputs.

Combinational Logic 2 – Equation

Equation should return 1 when the input equals 0, 6, or 12

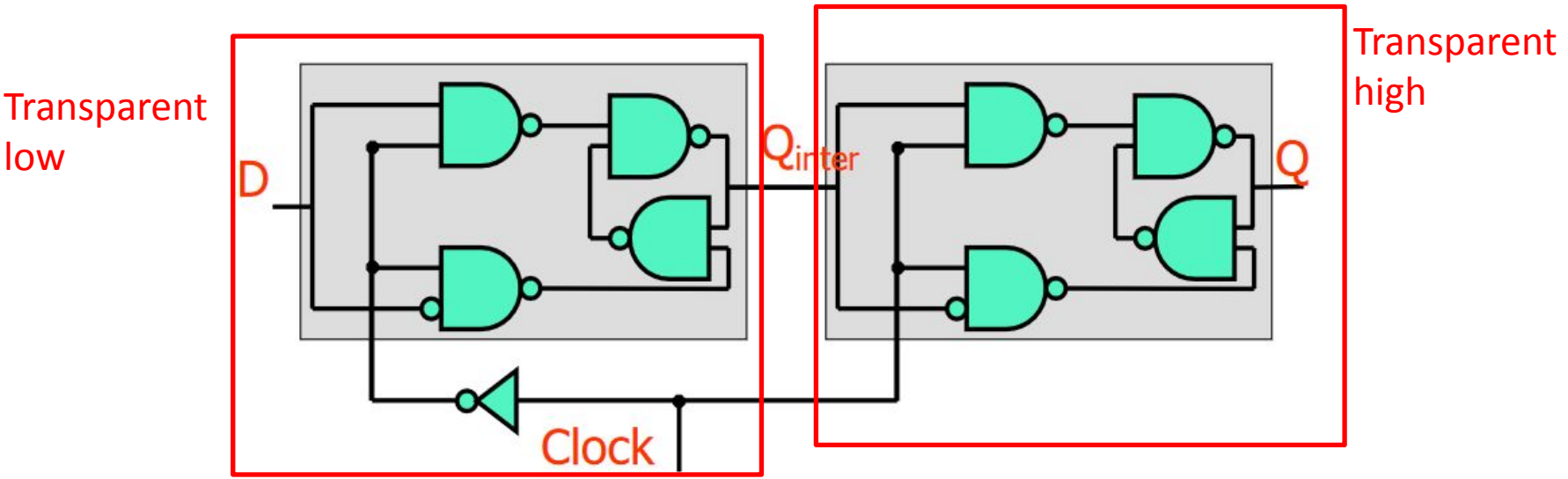
- 0000
- 0110
- 1100

$(\sim I_3 \,\&\, \sim I_2 \,\&\, \sim I_1 \,\&\, \sim I_0) \mid (\sim I_3 \,\&\, I_2 \,\&\, I_1 \,\&\, \sim I_0) \mid (I_3 \,\&\, I_2 \,\&\, \sim I_1 \,\&\, \sim I_0)$

Sequential Logic

A sequential circuit consists of logic gates whose outputs at any time are determined from both the present combination of inputs and previous output.

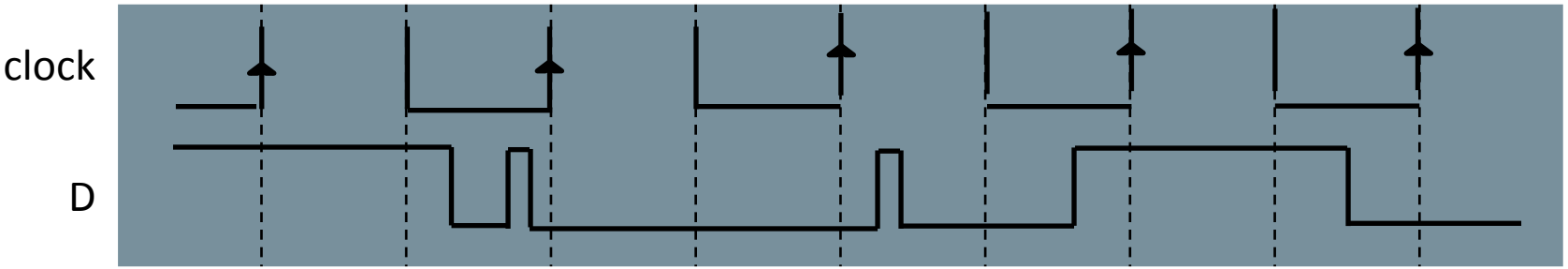
D Flip-Flop



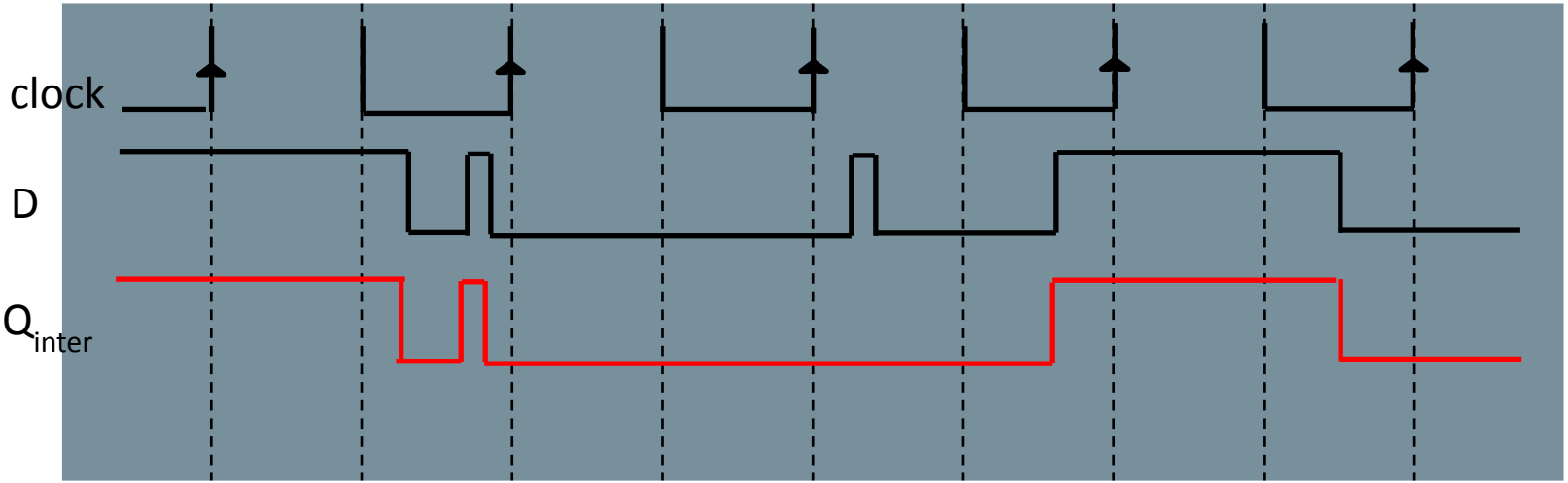
D Flip-Flop Practice

What is Q and Qinter for a D flip flop with the given clock and D signal?

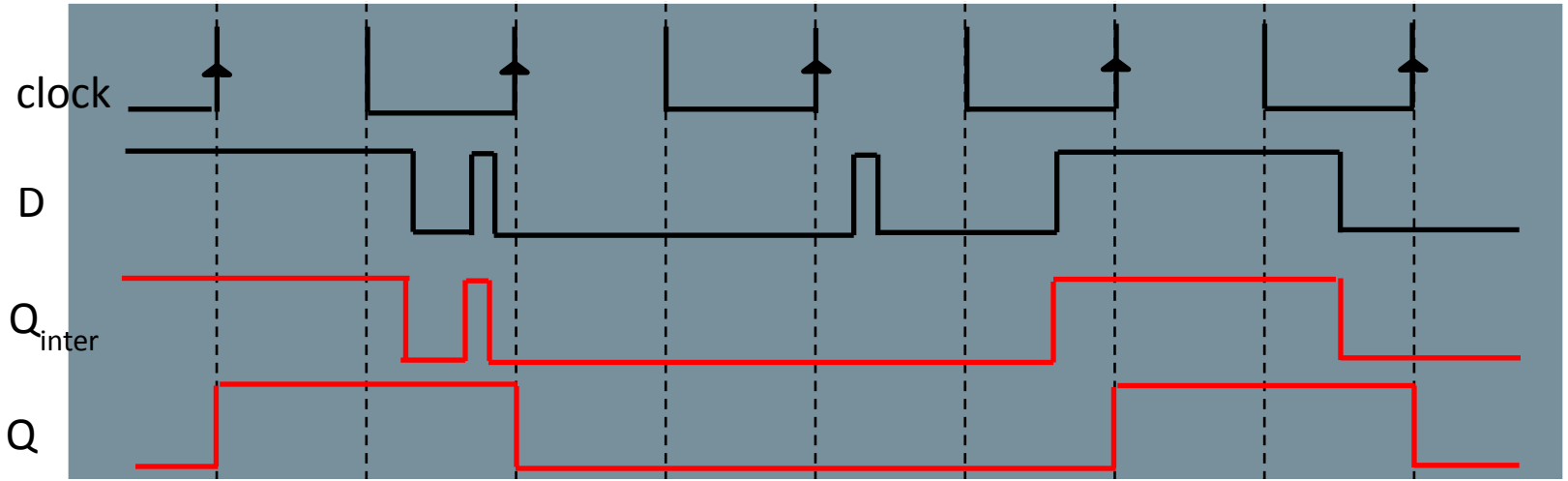
Assume that WE is 1



D Flip-Flop Practice



D Flip-Flop Practice



Control Signal Circuit

Your job in this question is to design part of the Decoder circuit for our Single Cycle LC4 implementation. Produce a circuit that takes as input the relevant bits from the current instruction and produce as output, the two-bit signal regInputMux.CTL

Hint: Since the output is two bits, you may want to make a circuit for each bit

Control Signal Circuit

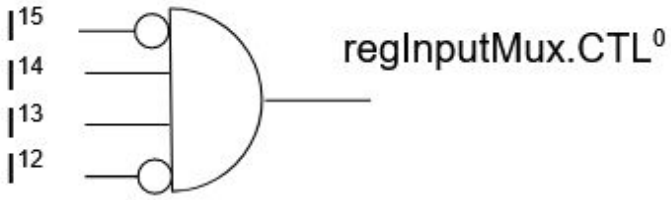
- Signal is two bits regInputMux.CTL1 and regInputMux.CTL0
- First, identify what inputs give what output signal
  - When is regInputMux.CTL0 equal to 1?
    - if and only if we want to read from memory
    - instruction is LDR
  - When is regInputMux.CTL1 equal to 1?
    - if and only if we want to store PC + 1 in register file
    - instruction is JSR, JSRR, or TRAP
- Make circuits that filters on the opcodes (and maybe sub-opcodes) for those instructions
  - opcode is the 4 most significant bits of the instruction
  - If instruction is not LDR, JSR, JSRR, or TRAP signal is 0

Control Signal Circuit

Keep it simple: PLA-like circuits

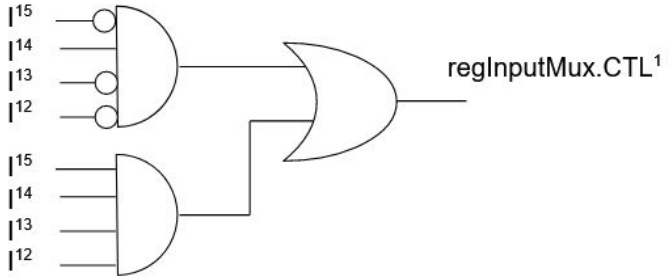
regInputMux.CTL0

- Instruction must be LDR for signal to be high
- LDR Opcode is 0110



regInputMux.CTL1

- Instruction must be JSR, JSRR or TRAP
- JSR & JSRR Opcode: 0100
- Trap Opcode: 1111



Decoding Signals: TRAP

Let's decode the TRAP instruction!

```
TRAP UIMM8
  | R7 = PC + 1; PC = (0x8000 | UIMM8); PSR [15] = 1
  | 1111 xxxx uuuu uuuu
```

rsMux.CTL	
rtMux.CTL	
rdMux.CTL	
regFile.WE	
ALUInputMux.CTL	
ALU.CTL	
regInputMux.CTL	
NZP.WE	
DATA.WE	
Privilege.CTL	

Decoding Signals: TRAP

Let's work through each component at a time, starting with setting R7. What registers / values do we need to retrieve?

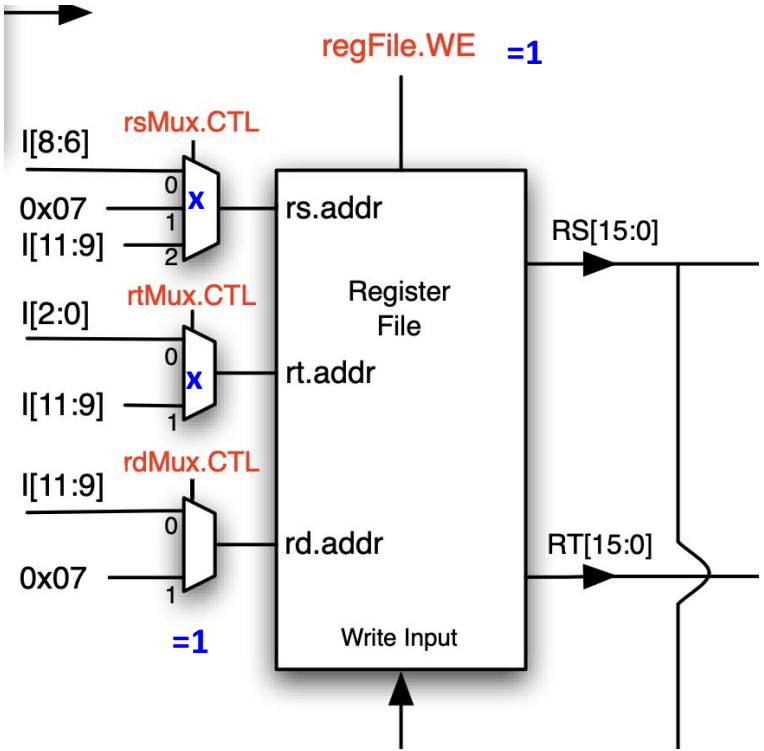
```
R7 = PC + 1; PC = (0x8000 | UIMM8); PSR [15] = 1
```

- Writing to register file
- R7 is our destination register
- We need to somehow retrieve PC + 1 and have it inputted to register file

Decoding Signals: TRAP

```
R7 = PC + 1;
```

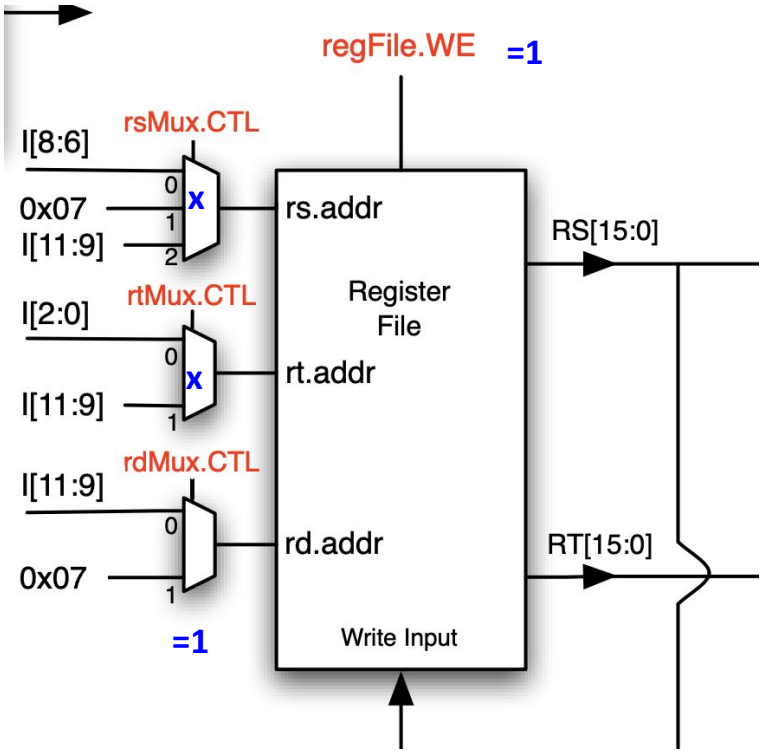
rsMux.CTL	
rtMux.CTL	
rdMux.CTL	
regFile.WE	
ALUInputMux.CTL	
ALU.CTL	
regInputMux.CTL	
NZP.WE	
DATA.WE	
Privilege.CTL	



Decoding Signals: TRAP

```
R7 = PC + 1;
```

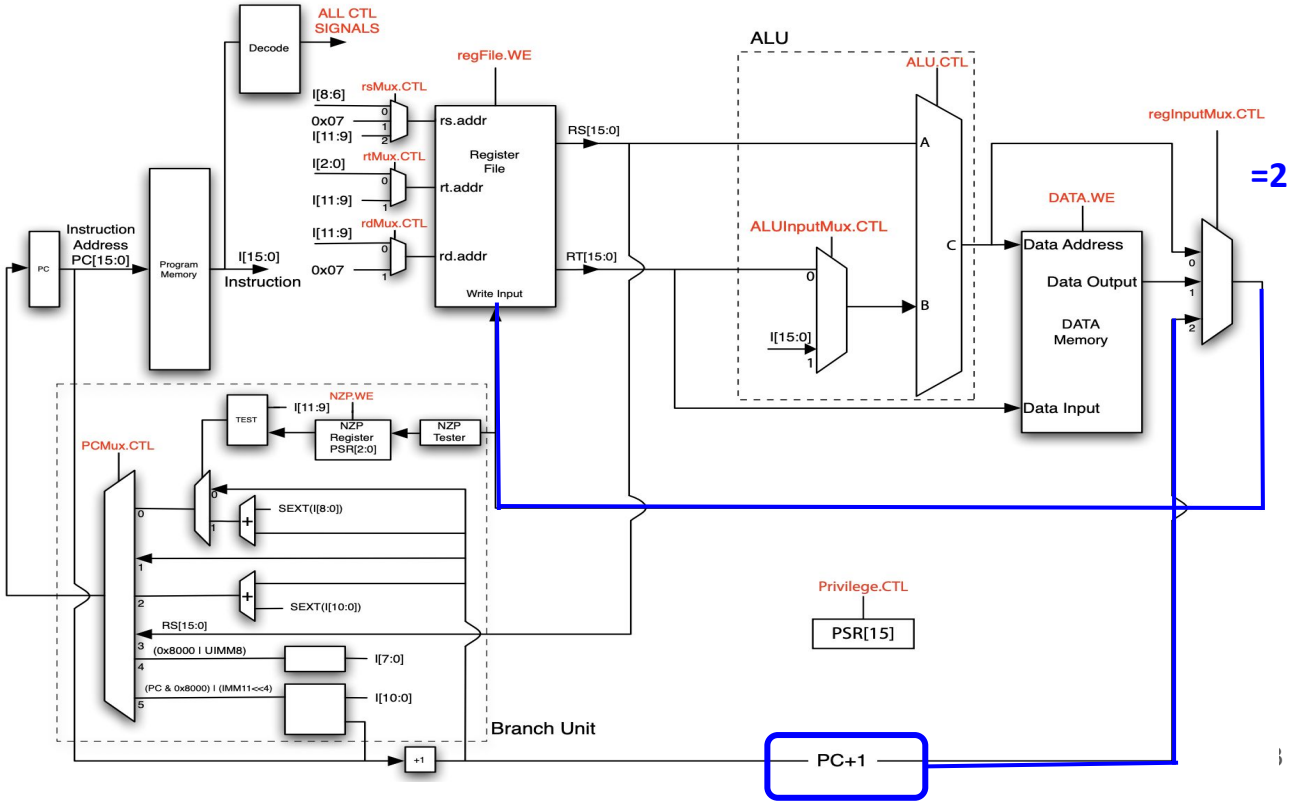
rsMux.CTL	X
rtMux.CTL	X
rdMux.CTL	1
regFile.WE	1
ALUInputMux.CTL	
ALU.CTL	
regInputMux.CTL	
NZP.WE	
DATA.WE	
Privilege.CTL	



Decoding Signals: TRAP

```
R7 = PC + 1;
```

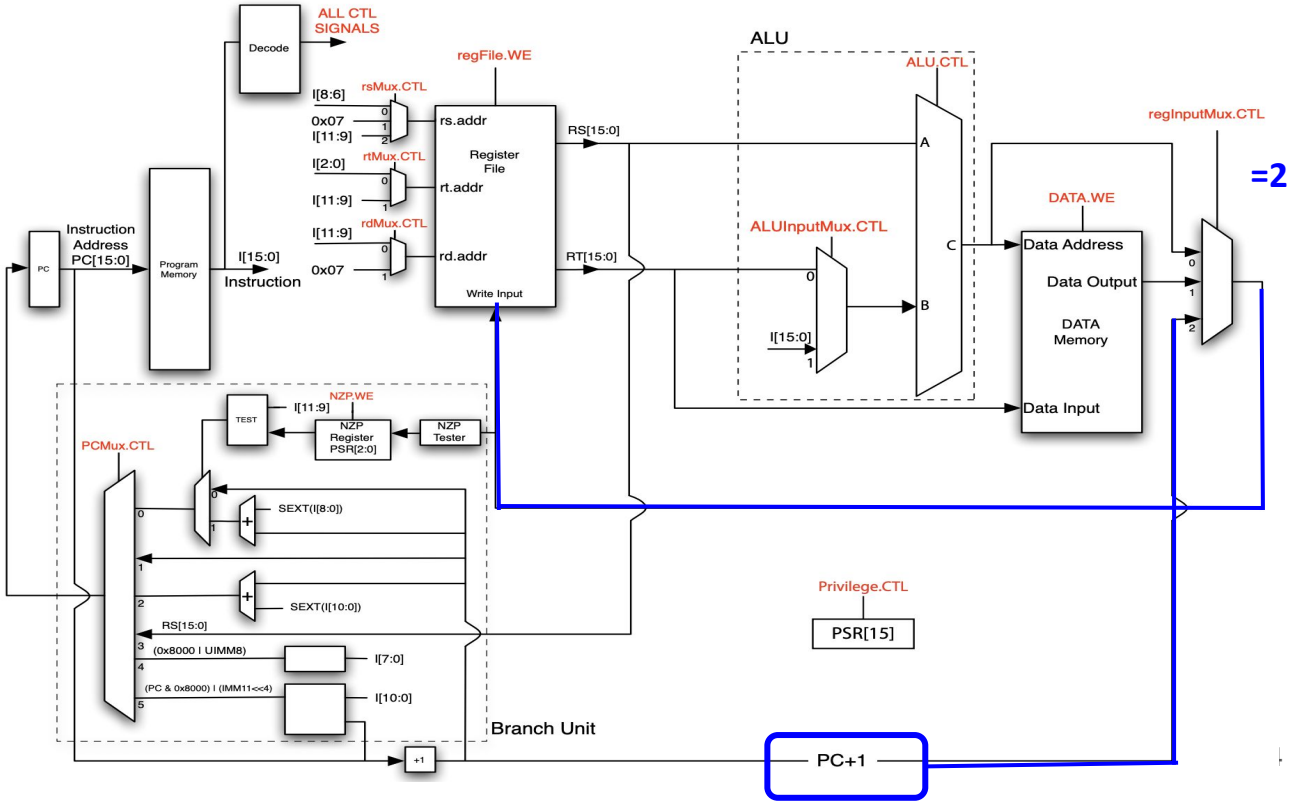
rsMux.CTL	X
rtMux.CTL	X
rdMux.CTL	1
regFile.WE	1
ALUInputMux.CTL	
ALU.CTL	
regInputMux.CTL	
NZP.WE	
DATA.WE	
Privilege.CTL	



Decoding Signals: TRAP

```
R7 = PC + 1;
```

rsMux.CTL	X
rtMux.CTL	X
rdMux.CTL	1
regFile.WE	1
ALUInputMux.CTL	
ALU.CTL	
regInputMux.CTL	2
NZP.WE	
DATA.WE	
Privilege.CTL	



Decoding Signals: TRAP

Now, we're done with the first part. Move on to the next (if there are any).

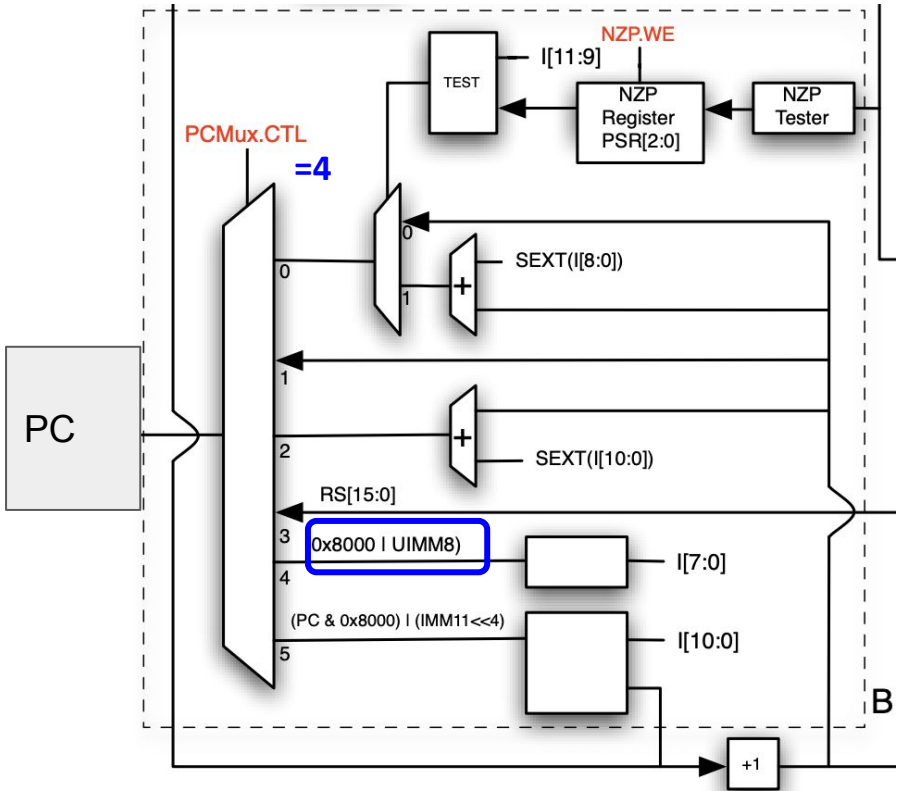
```
R7 = PC + 1; PC = (0x8000 | UIMM8); PSR [15] = 1
```

- Update PC to a value other than PC + 1
- Need to retrieve (0x8000 | UIMM8)

Decoding Signals: TRAP

rsMux.CTL	X
rtMux.CTL	X
rdMux.CTL	1
regFile.WE	1
ALUInputMux.CTL	
ALU.CTL	
regInputMux.CTL	2
PCMux.CTL	
NZP.WE	
DATA.WE	
Privilege.CTL	

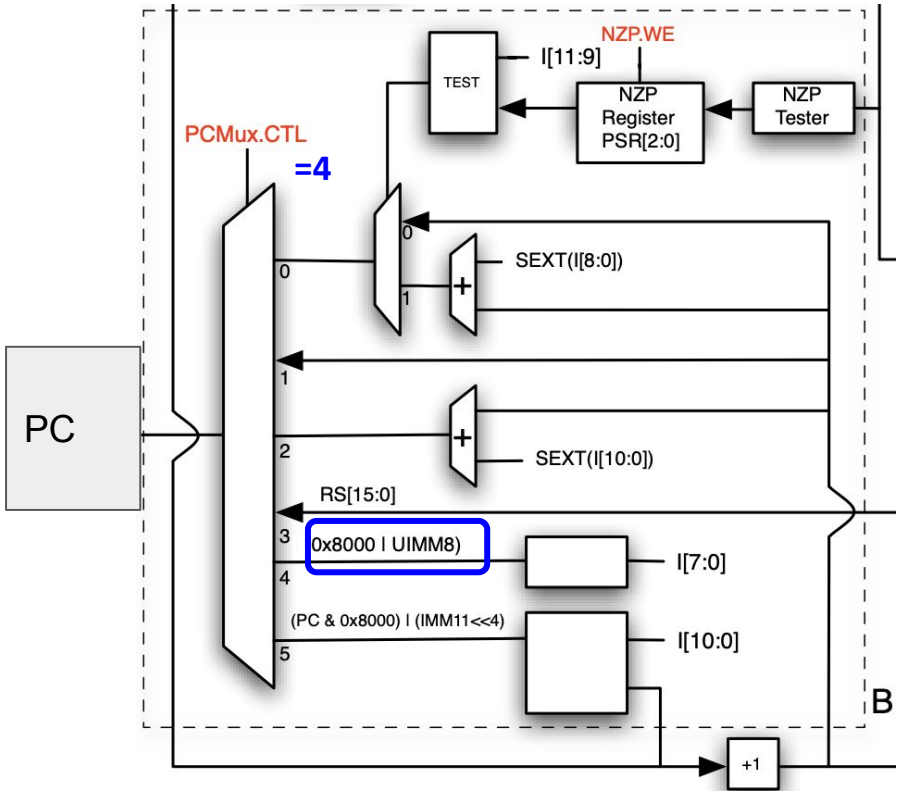
```
PC = (0x8000 | UIMM8);
```



Decoding Signals: TRAP

rsMux.CTL	X
rtMux.CTL	X
rdMux.CTL	1
regFile.WE	1
ALUInputMux.CTL	
ALU.CTL	
regInputMux.CTL	2
PCMux.CTL	4
NZP.WE	
DATA.WE	
Privilege.CTL	

```
PC = (0x8000 | UIMM8);
```



Decoding Signals: TRAP

Now, we're done with the second part. Move on to the next (if there are any).

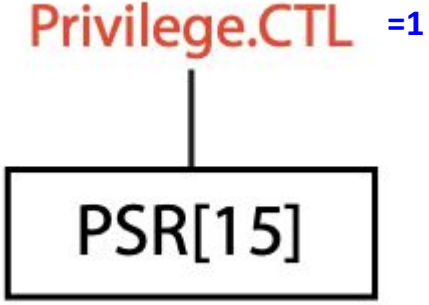
```
R7 = PC + 1; PC = (0x8000 | UIMM8); PSR [15] = 1
```

- Set the privilege bit to 1

Decoding Signals: TRAP

rsMux.CTL	X
rtMux.CTL	X
rdMux.CTL	1
regFile.WE	1
ALUInputMux.CTL	
ALU.CTL	
regInputMux.CTL	2
PCMux.CTL	3
NZP.WE	
DATA.WE	
Privilege.CTL	

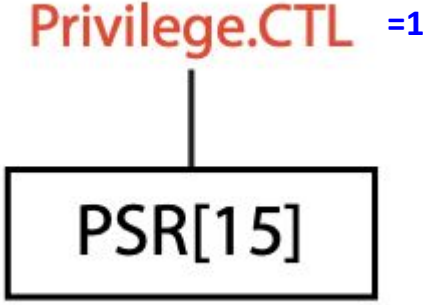
```
PSR [15] = 1
```



Decoding Signals: TRAP

rsMux.CTL	X
rtMux.CTL	X
rdMux.CTL	1
regFile.WE	1
ALUInputMux.CTL	
ALU.CTL	
regInputMux.CTL	2
PCMux.CTL	3
NZP.WE	
DATA.WE	
Privilege.CTL	1

```
PSR [15] = 1
```



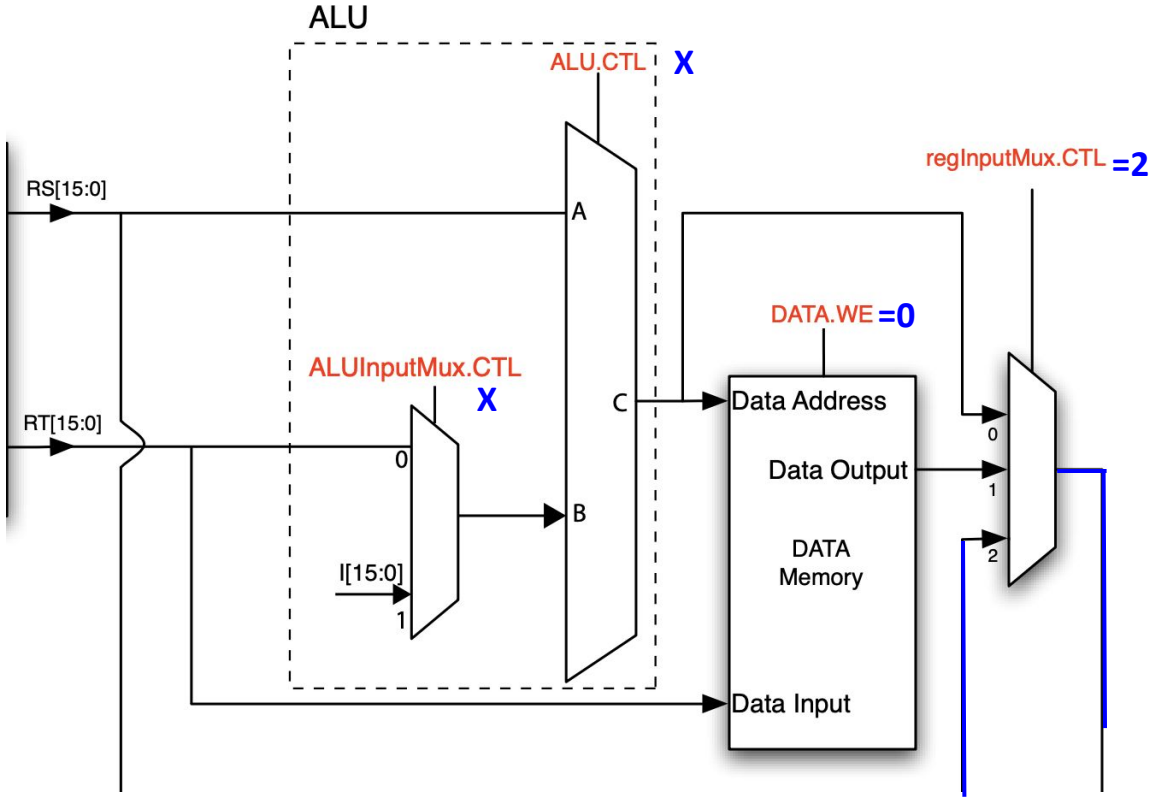
Decoding Signals: TRAP

We finished all that we needed to do! Now, fill out the rest of the control signals. Note that some are mandatory and others are not.

```
R7 = PC + 1; PC = (0x8000 | UIMM8); PSR [15] = 1
```

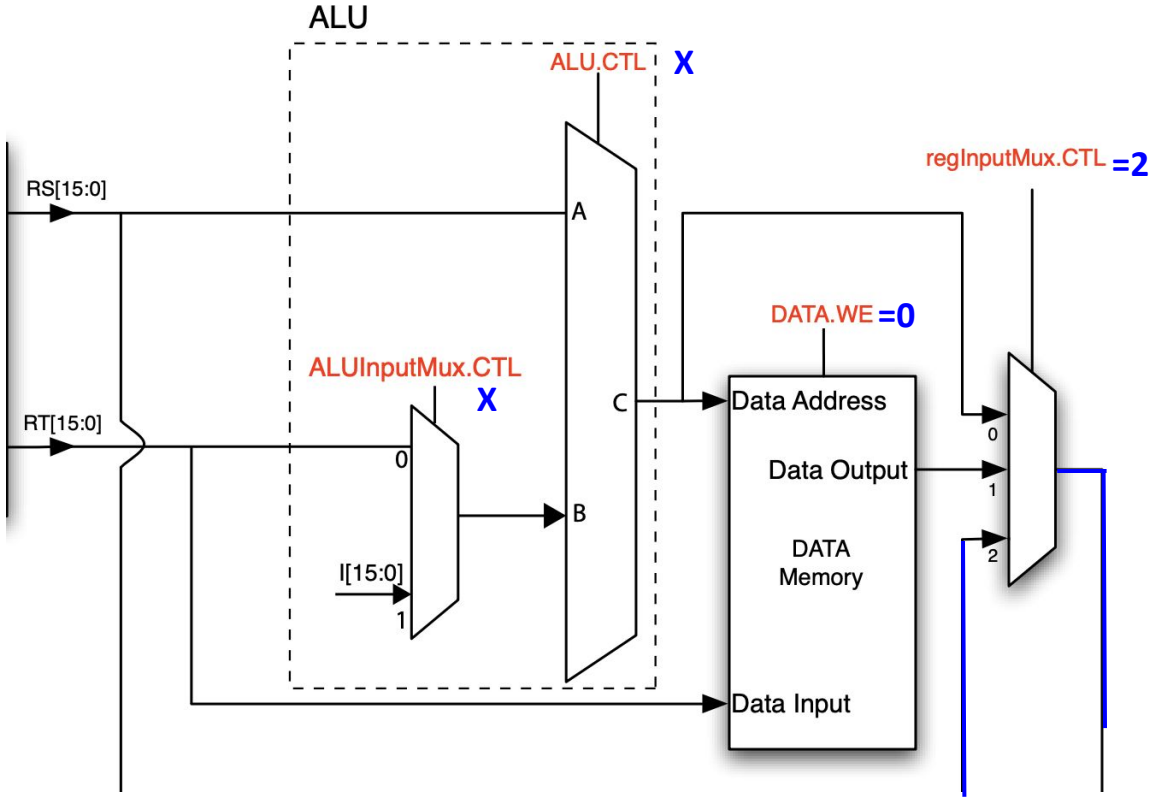
Decoding Signals: TRAP

rsMux.CTL	X
rtMux.CTL	X
rdMux.CTL	1
regFile.WE	1
ALUInputMux.CTL	
ALU.CTL	
regInputMux.CTL	2
PCMux.CTL	4
NZP.WE	
DATA.WE	
Privilege.CTL	1



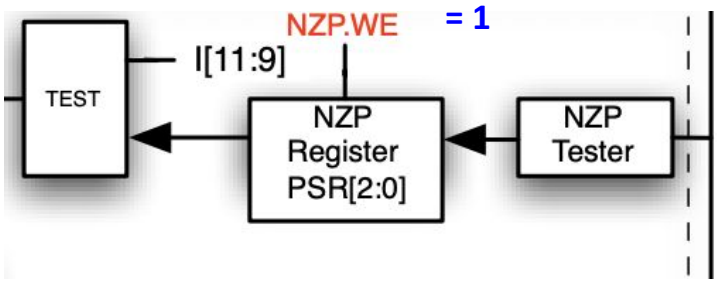
Decoding Signals: TRAP

rsMux.CTL	X
rtMux.CTL	X
rdMux.CTL	1
regFile.WE	1
ALUInputMux.CTL	X
ALU.CTL	X
regInputMux.CTL	2
PCMux.CTL	3
NZP.WE	
DATA.WE	0
Privilege.CTL	1



Decoding Signals: TRAP

rsMux.CTL	X
rtMux.CTL	X
rdMux.CTL	1
regFile.WE	1
ALUInputMux.CTL	X
ALU.CTL	X
regInputMux.CTL	2
PCMux.CTL	3
NZP.WE	
DATA.WE	0
Privilege.CTL	1



Decoding Signals: TRAP

rsMux.CTL	X
rtMux.CTL	X
rdMux.CTL	1
regFile.WE	1
ALUInputMux.CTL	X
ALU.CTL	X
regInputMux.CTL	2
PCMux.CTL	4
NZP.WE	1
DATA.WE	0
Privilege.CTL	1

Done!

LC4 Interpretation

Translate the given machine instructions to LC4 instructions, or vise versa

Address	Machine Instruction	Assembly Instruction
0x0000	1001 0110 0001 0000	CONST R3, #16
0x0001	1010 0000 0011 0011	
0x0002	1001 0010 0000 0001	
0x0003	0010 0001 0000 0000	
0x0004	0000 0100 0000 0011	
0x0005		SLL R1, R1, #1
0x0006		ADD R0, R0, #-1
0x0007		JMP #-5
0x0008		BRnzp #-1

LC4 Interpretation

Translate the given machine instructions to LC4 instructions, or vise versa

Address	Machine Instruction	Assembly Instruction
0x0000	1001011000010000	CONST R3, #16
0x0001	1010000000110011	MOD R0, R0, R3
0x0002	1001001000000001	
0x0003	0010000100000000	
0x0004	0000010000000011	
0x0005		SLL R1, R1, #1
0x0006		ADD R0, R0, #-1
0x0007		JMP #-5
0x0008		BRnzp #-1

LC4 Interpretation

Translate the given machine instructions to LC4 instructions, or vise versa

Address	Machine Instruction	Assembly Instruction
0x0000	1001011000010000	CONST R3, #16
0x0001	1010000000110011	MOD R0, R0, R3
0x0002	1001001000000001	CONST R1, #1
0x0003	0010000100000000	
0x0004	0000010000000011	
0x0005		SLL R1, R1, #1
0x0006		ADD R0, R0, #-1
0x0007		JMP #-5
0x0008		BRnzp #-1

LC4 Interpretation

Translate the given machine instructions to LC4 instructions, or vise versa

Address	Machine Instruction	Assembly Instruction
0x0000	1001011000010000	CONST R3, #16
0x0001	1010000000110011	MOD R0, R0, R3
0x0002	1001001000000001	CONST R1, #1
0x0003	0010000100000000	CMPI R0, #0
0x0004	0000010000000011	
0x0005		SLL R1, R1, #1
0x0006		ADD R0, R0, #-1
0x0007		JMP #-5
0x0008		BRnzp #-1

LC4 Interpretation

Translate the given machine instructions to LC4 instructions, or vise versa

Address	Machine Instruction	Assembly Instruction
0x0000	1001011000010000	CONST R3, #16
0x0001	1010000000110011	MOD R0, R0, R3
0x0002	1001001000000001	CONST R1, #1
0x0003	0010000100000000	CMPI R0, #0
0x0004	0000010000000011	BRz #3
0x0005		SLL R1, R1, #1
0x0006		ADD R0, R0, #-1
0x0007		JMP #-5
0x0008		BRnzp #-1

LC4 Interpretation

Translate the given machine instructions to LC4 instructions, or vise versa

Address	Machine Instruction	Assembly Instruction
0x0000	1001011000010000	CONST R3, #16
0x0001	1010000000110011	MOD R0, R0, R3
0x0002	1001001000000001	CONST R1, #1
0x0003	0010000100000000	CMPI R0, #0
0x0004	0000010000000011	BRz #3
0x0005	1010001001000001	SLL R1, R1, #1
0x0006		ADD R0, R0, #-1
0x0007		JMP #-5
0x0008		BRnzp #-1

LC4 Interpretation

Translate the given machine instructions to LC4 instructions, or vise versa

Address	Machine Instruction	Assembly Instruction
0x0000	1001011000010000	CONST R3, #16
0x0001	1010000000110011	MOD R0, R0, R3
0x0002	1001001000000001	CONST R1, #1
0x0003	0010000100000000	CMPI R0, #0
0x0004	0000010000000011	BRz #3
0x0005	1010001001000001	SLL R1, R1, #1
0x0006	0001000000111111	ADD R0, R0, #-1
0x0007		JMP #-5
0x0008		BRnzp #-1

LC4 Interpretation

Translate the given machine instructions to LC4 instructions, or vise versa

Address	Machine Instruction	Assembly Instruction
0x0000	1001011000010000	CONST R3, #16
0x0001	1010000000110011	MOD R0, R0, R3
0x0002	1001001000000001	CONST R1, #1
0x0003	0010000100000000	CMPI R0, #0
0x0004	0000010000000011	BRz #3
0x0005	1010001001000001	SLL R1, R1, #1
0x0006	0001000000111111	ADD R0, R0, #-1
0x0007	1100111111111011	JMP #-5
0x0008		BRnzp #-1

LC4 Interpretation

Translate the given machine instructions to LC4 instructions, or vise versa

Address	Machine Instruction	Assembly Instruction
0x0000	1001011000010000	CONST R3, #16
0x0001	1010000000110011	MOD R0, R0, R3
0x0002	1001001000000001	CONST R1, #1
0x0003	0010000100000000	CMPI R0, #0
0x0004	0000010000000011	BRz #3
0x0005	1010001001000001	SLL R1, R1, #1
0x0006	0001000000111111	ADD R0, R0, #-1
0x0007	1100111111111011	JMP #-5
0x0008	0000111111111111	BRnzp #-1

Writing LC4 ASM Code: Strategy

- Formulate your logic into pseudocode
- Create a dictionary - mapping LC4 registers to pseudocode variables
- Translate pseudocode lines to LC4 instructions, line-by-line
- Test the code
  - run a sample input, check the output against expected

Example: Palindrome

- Write an LC4 program that determines **whether a given string is a palindrome**. Note: a palindrome is a string that reads the same backwards and forwards (e.g. abba, aba, abdba, abcddcba).
- You can assume that the **starting address** of the string is given in **R0**, and the **string length** is given in **R1**.
- By the end of the program (END label), **R2** should contain **1** if the string is a **palindrome**, and **0** otherwise.
- By default, **R2 is initially set to 1**, assuming that an empty string is a palindrome
- Feel free to modify any/all registers, as long as R2 contains the correct result at the end of the program.

Step 1: Logic

Idea: two-pointer method. In a loop, iterate over the string from both ends, if two corresponding letters (e.g. first and last) are not equal, return 0. If the loop ends with all corresponding letters being equal, return 1.

Step 2: Pseudocode

```
int i = str_address (R0)
int j = length (R1) + str_address - 1 (j = end_address, last letter)
int result = 1 (R2)

while (i < j) {
      int first = string[i]
      int second = string[j]
      if (first != second) {
            result = 0
            return result;
      }
      i++
      j--
}

return result
```

Step 2: Dictionary (in assembly file)

```
;; Dictionary

;; R0 -> starting address
;; R1 -> length
;; R2 -> result
;; R3 -> j (end_address)
;; R4 -> first
;; R5 -> second
```

Step 3: Initialize the code address + end label

```
        .CODE
        .ADDR 0x0000
START

END
```

Step 3: translate pseudocode into assembly instructions

```
;; int j = str_address (R0) + length(R1) - 1

      ADD R3, R0, R1   ; j = str_address + length
      ADD R3, R3, #-1  ; j = j - 1
```

```
;; while (i < j)


LOOP
      ;; invert the logic, if (i >= j), exit loop to END

      CMP R0, R1
      BRzp END
```

Step 3: translate pseudocode into assembly instructions

```
;; int first = string[i]
LDR R4, R0, #0

;; int second = string[j]
LDR R5, R3, #0
```

```
;; if (first != second)
CMP R4, R5
BRnp NOT_PALINDROME
```

Step 3: translate pseudocode into assembly instructions

```
;; if (first != second)
CMP R4, R5
BRnp NOT_PALINDROME

;; i++
ADD R0, R0, #1

;; j--
ADD R3, R3, #-1
```

```
NOT_PALINDROME

    ;; result = 0
    CONST R2, #0
```

```
;; loop again
JMP LOOP
```

Full Code (.asm file)


```
      .CODE
      .ADDR 0x0000
START

;; int j = str_address (R0) + length(R1) - 1

      ADD R3, R0, R1   ; j = str_address + length
      ADD R3, R3, #-1  ; j = j - 1


;; while (i < j)


LOOP
      ;; invert the logic, if (i >= j), exit loop to END

      CMP R0, R1
      BRzp END

      ;; int first = string[i]
      LDR R4, R0, #0

      ;; int second = string[j]
      LDR R5, R3, #0

      ;; if (first != second)
      CMP R4, R5
      BRnp NOT_PALINDROME

      ;; i++
      ADD R0, R0, #1

      ;; j--
      ADD R3, R3, #-1

      ;; loop again
      JMP LOOP


NOT_PALINDROME

      ;; result = 0
      CONST R2, #0

END
```

LC4 Trace

Simulate the assembly program by filling in this table Fill in the NZP entries with N, Z or P. Always fill in NZP & PC. For other registers, you only need to fill in the values of registers that have changed from the previous cycle

addr	Instruction
0	CONST R3, #16
1	MOD R0, R0, R3
2	CONST R1, #1
3	CMPI R0, #0
4	BRz #3
5	SLL R1, R1, #1
6	ADD R0, R0, #-1
7	JMP #-5
8	BRnzp #-1

Cycle	0	1	2	3	4	5	6	7	8	9	10	11
PC	0											
NZP	P											
R0	33											
R1	0											
R2	0											
R3	0											
R4	0											
R5	0											
R6	0											
R7	0											

LC4 Trace

addr	Instruction
0	CONST R3, #16
1	MOD R0, R0, R3
2	CONST R1, #1
3	CMPI R0, #0
4	BRz #3
5	SLL R1, R1, #1
6	ADD R0, R0, #-1
7	JMP #-5
8	BRnzp #-1

Cycle	0	1	2	3	4	5	6	7	8	9	10	11
PC	0	1										
NZP	P	P										
R0	33											
R1	0											
R2	0											
R3	0	16										
R4	0											
R5	0											
R6	0											
R7	0											

LC4 Trace

addr	Instruction
0	CONST R3, #16
1	MOD R0, R0, R3
2	CONST R1, #1
3	CMPI R0, #0
4	BRz #3
5	SLL R1, R1, #1
6	ADD R0, R0, #-1
7	JMP #-5
8	BRnzp #-1

Cycle	0	1	2	3	4	5	6	7	8	9	10	11
PC	0	1	2									
NZP	P	P	P									
R0	33											
R1	0											
R2	0											
R3	0	16	1									
R4	0											
R5	0											
R6	0											
R7	0											

LC4 Trace

addr	Instruction
0	CONST R3, #16
1	MOD R0, R0, R3
2	CONST R1, #1
3	CMPI R0, #0
4	BRz #3
5	SLL R1, R1, #1
6	ADD R0, R0, #-1
7	JMP #-5
8	BRnzp #-1

Cycle	0	1	2	3	4	5	6	7	8	9	10	11
PC	0	1	2	3								
NZP	P	P	P	P								
R0	33											
R1	0			1								
R2	0											
R3	0	16	1									
R4	0											
R5	0											
R6	0											
R7	0											

LC4 Trace

addr	Instruction
0	CONST R3, #16
1	MOD R0, R0, R3
2	CONST R1, #1
3	CMPI R0, #0
4	BRz #3
5	SLL R1, R1, #1
6	ADD R0, R0, #-1
7	JMP #-5
8	BRnzp #-1

Cycle	0	1	2	3	4	5	6	7	8	9	10	11
PC	0	1	2	3	4							
NZP	P	P	P	P	P							
R0	33											
R1	0			1								
R2	0											
R3	0	16	1									
R4	0											
R5	0											
R6	0											
R7	0											

LC4 Trace

addr	Instruction
0	CONST R3, #16
1	MOD R0, R0, R3
2	CONST R1, #1
3	CMPI R0, #0
4	BRz #3
5	SLL R1, R1, #1
6	ADD R0, R0, #-1
7	JMP #-5
8	BRnzp #-1

Cycle	0	1	2	3	4	5	6	7	8	9	10	11
PC	0	1	2	3	4	5						
NZP	P	P	P	P	P	P						
R0	33											
R1	0			1								
R2	0											
R3	0	16	1									
R4	0											
R5	0											
R6	0											
R7	0											

LC4 Trace

addr	Instruction
0	CONST R3, #16
1	MOD R0, R0, R3
2	CONST R1, #1
3	CMPI R0, #0
4	BRz #3
5	SLL R1, R1, #1
6	ADD R0, R0, #-1
7	JMP #-5
8	BRnzp #-1

Cycle	0	1	2	3	4	5	6	7	8	9	10	11
PC	0	1	2	3	4	5	6					
NZP	P	P	P	P	P	P	P					
R0	33											
R1	0			1			2					
R2	0											
R3	0	16	1									
R4	0											
R5	0											
R6	0											
R7	0											

LC4 Trace

addr	Instruction
0	CONST R3, #16
1	MOD R0, R0, R3
2	CONST R1, #1
3	CMPI R0, #0
4	BRz #3
5	SLL R1, R1, #1
6	ADD R0, R0, #-1
7	JMP #-5
8	BRnzp #-1

Cycle	0	1	2	3	4	5	6	7	8	9	10	11
PC	0	1	2	3	4	5	6	7				
NZP	P	P	P	P	P	P	P	Z				
R0	33							0				
R1	0			1			2					
R2	0											
R3	0	16	1									
R4	0											
R5	0											
R6	0											
R7	0											

LC4 Trace

addr	Instruction
0	CONST R3, #16
1	MOD R0, R0, R3
2	CONST R1, #1
3	CMPI R0, #0
4	BRz #3
5	SLL R1, R1, #1
6	ADD R0, R0, #-1
7	JMP #-5
8	BRnzp #-1

Cycle	0	1	2	3	4	5	6	7	8	9	10	11
PC	0	1	2	3	4	5	6	7	3			
NZP	P	P	P	P	P	P	P	Z	Z			
R0	33							0				
R1	0			1			2					
R2	0											
R3	0	16	1									
R4	0											
R5	0											
R6	0											
R7	0											

LC4 Trace

addr	Instruction
0	CONST R3, #16
1	MOD R0, R0, R3
2	CONST R1, #1
3	CMPI R0, #0
4	BRz #3
5	SLL R1, R1, #1
6	ADD R0, R0, #-1
7	JMP #-5
8	BRnzp #-1

Cycle	0	1	2	3	4	5	6	7	8	9	10	11
PC	0	1	2	3	4	5	6	7	3	4		
NZP	P	P	P	P	P	P	P	Z	Z	Z		
R0	33							0				
R1	0			1			2					
R2	0											
R3	0	16	1									
R4	0											
R5	0											
R6	0											
R7	0											

LC4 Trace

addr	Instruction
0	CONST R3, #16
1	MOD R0, R0, R3
2	CONST R1, #1
3	CMPI R0, #0
4	BRz #3
5	SLL R1, R1, #1
6	ADD R0, R0, #-1
7	JMP #-5
8	BRnzp #-1

Cycle	0	1	2	3	4	5	6	7	8	9	10	11
PC	0	1	2	3	4	5	6	7	3	4	8	
NZP	P	P	P	P	P	P	P	Z	Z	Z	Z	
R0	33							0				
R1	0			1			2					
R2	0											
R3	0	16	1									
R4	0											
R5	0											
R6	0											
R7	0											

LC4 Trace

addr	Instruction
0	CONST R3, #16
1	MOD R0, R0, R3
2	CONST R1, #1
3	CMPI R0, #0
4	BRz #3
5	SLL R1, R1, #1
6	ADD R0, R0, #-1
7	JMP #-5
8	BRnzp #-1

Cycle	0	1	2	3	4	5	6	7	8	9	10	11
PC	0	1	2	3	4	5	6	7	3	4	8	8
NZP	P	P	P	P	P	P	P	Z	Z	Z	Z	Z
R0	33							0				
R1	0			1			2					
R2	0											
R3	0	16	1									
R4	0											
R5	0											
R6	0											
R7	0											

That's all we have for today!

Reminders:

- **MIDTERM is on Wednesday during lecture time in CHEM 102!**
  You **must** show up in person to take the exam. Please remember to wear a mask.
  More details/practice are on Ed and the course website.
- TA-lead recitations will take place on
  - Tuesdays 6:30-8:00pm in Moore 100A
  - Wednesday 12:00-1:30pm in Moore 100C
- Check the course website for OH times